SAMS

21天自学教程

——Visual C++

[美] Namir Clement Shammas 著

沈官林 尚 杰 钟向群 译

沈官林 天 峰 校

清华大学出版社

(京) 新登字 158 号

21 天自学教程——Visual C+ +

Teach Yourself Visual C+ + in 21 Days

Namir Clement Shammas

Authorized translation from the English language edition published by Sams.
Copyright © 1993 by Sams publishing.
All rights reserved. No part of this book may be reproduced or transmitted in any form or by any means, electronic or mechanical, including photocopying, recording or by any information storage retrieval system, without permission in writing from the publisher.
Chinese language edition published by Tsinghua University Press.
Copyright © 1994 by Tsinghua University Press.
本书英文版由 Prentice Hall 出版社属下的 Sams 计算机图书出版公司于 1993 年出版。版权为 Sams 所有。Sams 将本书的中文版专有出版权授予清华大学出版社。未经出版者书面允许,不得以任何方式复制或抄袭本书的内容。

本书封面贴有清华大学出版社激光防伪标签,无标签者不得销售。

图书在版编目(CIP)数据

21 天自学教程——Visual C+ + /(美)沙姆马斯(Shammas, N. C)著;沈官林等译. —北京:清华大学出版社,1994. 8

书名原文: Teach Yourself Visual C+ + in 21 Days

ISBN 7-302-01629-1

Ⅰ. 21… Ⅱ. ①沙… ②沈… Ⅲ. C 语言-教材 Ⅳ. TP312-43

中国版本图书馆 CIP 数据核字(94)第 11032 号

出版者: 清华大学出版社(北京清华大学校内,邮编 100084)
责任编辑: 姜峰
印刷者: 国防工业出版社印刷厂
发行者: 新华书店总店北京科技发行所
开　本: 787×1092　1/16　印张: 35　字数: 848 千字
版　次: 1994 年 11 月第 1 版　1994 年 11 月第 1 次印刷
书　号: ISBN 7-302-01629-1/TP・691
印　数: 0001—5000
定　价: 48.00 元

如何使用本书

《21 天自学教程——Visual C+ + 》的主要目的有两个: 教你用 C+ + 编程以及教你用 Visual C+ + 建立 Windows 应用程序。这里并不要求你以前编过程序, 但是知道如何使用其他语言——如 Basic 或 Pascal——编程的话当然会有所帮助。本书也并不是为一窍不通的人来写的, 因为学习 C+ + 编程和用 C+ + 编写 Windows 应用程序毕竟是两件非同寻常之事!

本书包括 21 章——每个学习日一章。由于本书既定目标的原因, 进度安排得比较快。每一章都包括[问与答]部分、[测验]部分和[练习]部分。每周过后都有一次[该周内容复习]。

本书的特殊段落

《21 天自学教程——Visual C+ + 》中插入了一些特殊的段落以便在学习过程中对你进行指导。[语法]段落帮助你掌握特定的 C+ + 原理。每一段落中都有实际示例和对 C+ + 概念的彻底阐述。要想对[语法]段落有一感性认识, 请看下面的例子。(不要试图弄懂其内容, 你还没有开始第 1 天的学习呢。)

语　法

一个选择的 if 语句

只有一个选择的 if 语句的一般语法如下:

```
if (condition)
    statement;
```

这是一条执行语句的情况, 如果是一连串的执行语句, 则应为:

```
if (tested_condition) {
   < sequence of statements>
}
```

示　例:

```
if (numberOfLines < 0)
    numberOfLines = 0;
if (height - 54 < 3) {
    area = length * width;
    volume = area * height;
}
```

本书的其他特殊段落还有[注意]、[警告]以及[要与不要]。

注　意　[注意]段落提供正在讨论的主题的附加信息。

警　告	[警告]段落提醒你与正在讨论的主题有关的一些有害的东西。

[要与不要]段落告诉你要做什么和不要做什么的诀窍。例如:

要读一下本小节的其余部分。它对每天结束时的作业部分做了说明。
不要跳过任何测验题和练习题。如果能完成当天的作业,就可学习新内容了。

书中提供了大量的带有注解的例子以帮助你学习编程。每天结束时都有一[问与答]小节,其中包含对与当天内容有关的一些常见问题的解答;还有[作业],其中包括测验和练习。

但是,要想学习 Visual C+ + 光靠读这本书是不行的,如要成为程序员,就得写程序。每一套测验题后都有一套练习题,建议你每道练习题都试做一下。编写 Visual C+ + 代码是学习 Visual C+ + 程序设计语言的最好方法。

本书中所用的约定

本书使用各种不同字体以帮助你区别 Visual C+ + 代码与正常英语,并帮助标识重要的概念。书中提及的实际的 Visual C+ + 代码是用小号英文字体排版的;占位符(Placeholders)——用来表示在代码内实际键入内容的术语——是用斜体字来排版的;新的或重要的词语用黑体排版。

新　词

有些关键新词在[新词]段落中突出显示。本书后面有一词汇表,需要的时候可以查阅一些易混淆的计算机名词。

衷心希望本书能使你向程序设计世界迈出大胆的一步。这是一个令人激动的世界,它可以使你能更自如地控制计算机。祝你编程顺利!

原 书 致 谢

感谢 Sams 出版公司的众多朋友鼓励我并和我一起完成此书。感谢出版人 Richard Swadley 和 Jordan Gold 的大力支持。非常感谢组稿主任 Stacy Hiquet 和我一起对本书进行了构想。特别要感谢开发编辑 Dean Miller, 他全力以赴保证了紧凑的时间安排并合理调整了书的容量。还要感谢技术审阅 Greg Guntle 的富有价值的意见。最后, 再次感谢 Sams 出版公司所有参与此书出版的人。

作 者 简 介

Namir Clement Shammas, 软件工程师、面向对象程序设计专家。他为一流水平的计算机杂志写过许多文章并负责过许多书的计算机程序设计部分, 包括 Turbo Pascal 6 Object-Oriented Programming, Windows Programmer's Guide to ObjectWindows Library 以及 Teach Yourself QBasic in 21 Days。

目录

第 1 周概要 …… 1
第1天 入门 …… 3
1.1 C+ + 程序基础 …… 3
1.2 装载 Visual C+ + 工作台 …… 3
1.3 Visual C+ + 工作台概观 …… 4
1.4 File 菜单项 …… 4
New 选择项 …… 4
Open 选择项 …… 4
Close 选择项 …… 4
Save 选择项 …… 5
Save As 选择项 …… 5
Save All 选择项 …… 6
Print 选择项 …… 6
Page Setup 选择项 …… 6
Exit 选择项 …… 6
Most Recent Files 选择项 …… 6
1.5 Edit 菜单项 …… 6
Undo 选择项 …… 6
Redo 选择项 …… 6
Cut 选择项 …… 6
Copy 选择项 …… 7
Paste 选择项 …… 7
Delete 选择项 …… 7
Find 选择项 …… 7
Replace 选择项 …… 8
Find Matching Brace 选择项 …… 8
Read Only 选择项 …… 8
1.6 View 菜单项 …… 9
Line 选择项 …… 9
Mixed Source/Asm 选择项 …… 9
Next Error 选择项 …… 9
Previous Error 选择项 …… 9
Toggle Bookmark 选择项 …… 9
Next Bookmark 选择项 …… 9
Previous Bookmark 选择项 …… 9
Clear All Bookmarks 选择项 …… 9
Toolbar 选择项 …… 9
Status Bar 选择项 …… 10
Syntax Coloring 选择项 …… 10
1.7 Project 菜单项 …… 10
AppWizard 选择项 …… 10
New 选择项 …… 10
Open 选择项 …… 10
Edit 选择项 …… 10
Close 选择项 …… 10
Compile File 选择项 …… 10
Build 选择项 …… 11
Rebuild All 选择项 …… 11
Stop Build 选择项 …… 11
Execute 选择项 …… 11
Scan Dependencies 选择项 …… 11
Scan All Dependencies 选择项 …… 11
Load Workspace 选择项 …… 11
Save Workspace 选择项 …… 11
Most Recent Projects 选择项 …… 11
1.8 Browse 菜单 …… 12
Go to Definition 选择项 …… 12
Go to Reference 选择项 …… 12
Next 选择项 …… 12
Previous 选择项 …… 12
Pop Context 选择项 …… 12
Open 选择项 …… 12
ClassWizard 选择项 …… 12
1.9 Debug 菜单 …… 12
1.10 Tools 菜单 …… 13
1.11 Options 菜单 …… 13
Project 选择项 …… 13
Debug 选择项 …… 13
Directories 选择项 …… 13
Editor 选择项 …… 13
Workspace 选择项 …… 13
Tools 选择项 …… 13
Color 选择项 …… 13
Fonts 选择项 …… 14
1.12 Window 菜单项 …… 14
1.13 Help 菜单 …… 14
1.14 QuickWin 应用程序 …… 14
1.15 你的第一个 C+ + 程序 …… 15

1.16 退出工作台 …… 18
小结 …… 18
问与答 …… 19
作业 …… 19
测验 …… 19
练习 …… 20

第2天 C++ 程序的组成 …… 21
2.1 Visual C++ 中的预定义数据类型 … 21
2.2 在 Visual C++ 中命名标识符 …… 22
2.3 # include 指令 …… 23
2.4 声明变量 …… 23
2.5 声明常量 …… 25
使用基于宏的常量 …… 26
使用正规常量 …… 27
2.6 声明函数及定义其原型 …… 28
2.7 函数中的局部变量 …… 30
2.8 函数中的静态变量 …… 32
2.9 嵌入函数 …… 33
2.10 从函数中退出 …… 35
2.11 缺省变元 …… 35
2.12 函数重载 …… 37
小结 …… 39
问与答 …… 40
作业 …… 40
测验 …… 40
练习 …… 42

第3天 运算符和表达式 …… 43
3.1 算术运算符 …… 43
3.2 算术表达式 …… 46
3.3 增量运算符 …… 46
3.4 赋值运算符 …… 48
3.5 sizeof 运算符 …… 51
3.6 类型强制 …… 53
3.7 关系和逻辑运算 …… 56
3.8 布尔表达式 …… 57
3.9 位操作运算符 …… 60
3.10 逗号运算符 …… 62
运算符优先级和求值方向 …… 62
小结 …… 64
问与答 …… 65
作业 …… 65
测验 …… 65
练习 …… 66

第4天 管理 I/O …… 67
4.1 格式化流输出 …… 67
4.2 流输入 …… 69
4.3 printf 函数 …… 71
小结 …… 74
问与答 …… 75
作业 …… 75
测验 …… 75
练习 …… 75

第5天 决策机构 …… 76
5.1 有一个选择的 if 语句 …… 76
5.2 有两个选择分支的 if-else 语句 …… 78
if 语句的潜在问题 …… 80
5.3 有多个选择的 if-else 语句 …… 80
5.4 switch 语句 …… 83
5.5 嵌套的决策机构 …… 86
小结 …… 88
问与答 …… 89
作业 …… 90
测验 …… 90
练习 …… 91

第6天 循环 …… 92
6.1 for 循环 …… 92
6.2 使用 for 循环的开循环 …… 95
6.3 do-while 循环 …… 97
6.4 while 循环 …… 99
6.5 跳过循环迭代 …… 101
6.6 退出循环 …… 103
6.7 嵌套循环 …… 103
小结 …… 105
问与答 …… 106
作业 …… 107
测验 …… 107
练习 …… 108

第7天 数组 …… 109
7.1 声明一维数组 …… 109
7.2 使用一维数组 …… 110
7.3 初始化一维数组 …… 112
7.4 函数中的数组参数 …… 115
7.5 数组排序 …… 118
7.6 数组查找 …… 122

7.7 多维数组 …… 127
7.8 初始化多维数组 …… 131
7.9 多维数组参数 …… 132
小结 …… 135
问与答 …… 136
作业 …… 137
测验 …… 137
练习 …… 138
第1周内容复习 …… 139

第2周概要 …… 141

第8天 用户定义类型和指针 …… 143
8.1 C++ 中的类型定义 …… 143
8.2 枚举数据类型 …… 144
8.3 结构 …… 147
8.4 联合 …… 151
8.5 引用变量 …… 151
8.6 指针概述 …… 153
8.7 指向现有变量的指针 …… 153
8.8 指向数组的指针 …… 155
指针增量/减量的方法 …… 158
8.9 指向结构的指针 …… 159
8.10 指针和动态内存 …… 162
8.11 远指针 …… 165
小结 …… 166
问与答 …… 167
作业 …… 167
测验 …… 168
练习 …… 168

第9天 字符串 …… 169
9.1 C++ 字符串 …… 169
9.2 串输入 …… 170
9.3 使用 STRING.H 函数库 …… 170
9.4 串赋值 …… 171
9.5 串的长度 …… 172
9.6 串的连接 …… 173
9.7 串比较 …… 176
9.8 串的转换 …… 180
9.9 串逆置 …… 181
9.10 字符定位 …… 183
9.11 串定位 …… 185
小结 …… 189
问与答 …… 190
作业 …… 190
测验 …… 191
练习 …… 191

第10天 高级函数参数 …… 192
10.1 传递数组变元 …… 192
10.2 使用字符串作变元 …… 195
10.3 使用结构变元 …… 197
10.4 由引用传递变元 …… 198
10.5 由引用传递结构 …… 199
10.6 用指针传递结构 …… 200
10.7 递归函数 …… 202
10.8 传递指向动态结构的指针 …… 204
10.9 指向函数的指针 …… 207
小结 …… 215
问与答 …… 216
作业 …… 216
测验 …… 216
练习 …… 218

第11天 面向对象的程序设计和 C++ 类 …… 219
11.1 面向对象程序设计的基本知识 …… 219
11.2 类和对象 …… 219
11.3 消息和方法 …… 220
11.4 继承 …… 220
11.5 多态性 …… 220
11.6 声明基类 …… 221
11.7 类的各成分 …… 221
11.8 构造函数 …… 224
11.9 析构函数 …… 226
11.10 构造函数和析构函数举例 …… 227
11.11 声明类层次结构 …… 229
11.12 虚函数 …… 233
11.13 虚函数规则 …… 237
11.14 友元函数 …… 238
11.15 运算符和友元运算符 …… 241
小结 …… 245
问与答 …… 246
作业 …… 247
测验 …… 247
练习 …… 248

第12天 基本流文件 I/O …… 249

12.1 C++ 流库 …… 249
12.2 通用流 I/O 函数 …… 249
12.3 顺序文本流 I/O …… 251
12.4 顺序二进制文件流 I/O …… 255
12.5 随机存取文件流 I/O …… 261
小结 …… 265
问与答 …… 266
作业 …… 266
测验 …… 266
练习 …… 267

第13天 用 MFC 库编写 Windows 程序 …… 268
13.1 常用的 Windows 数据类型 …… 268
13.2 MFC 层次结构 …… 270
CObject 类 …… 270
CMenu 类 …… 270
文档体系结构(Document Architecture)子层 …… 270
窗口框架(Frames Windows)子层 …… 270
控制栏(Control Bars)子层 …… 270
视窗(Views)子层 …… 271
对话框(Dialog Boxes)子层 …… 271
控制(Control)子层 …… 271
绘图(Graphical Drawing)子层 …… 271
绘图对象(Graphical Drawing Objects)子层 …… 271
其他 MFC 子层 …… 271
MFC 宏 …… 272
13.3 Windows API 函数 …… 272
13.4 调用 Windows API 函数 …… 274
13.5 Windows 消息 …… 276
13.6 对消息进行响应 …… 279
13.7 发送消息 …… 280
13.8 用户定义消息 …… 281
小结 …… 282
问与答 …… 282
作业 …… 283
测验 …… 283
练习 …… 283

第14天 建立基本的 MFC 应用程序 …… 284
14.1 建立一个小的 MFC 应用程序 …… 284
14.2 扩充窗口操作 …… 288
14.3 响应菜单选择 …… 293
14.4 建立多个实例 …… 299
14.5 关闭窗口 …… 304
小结 …… 307
问与答 …… 308
作业 …… 308
测验 …… 308
练习 …… 309
第 2 周内容复习 …… 310

第 3 周概要 …… 313

第15天 基本窗口 …… 315
15.1 建立只读文本窗口 …… 315
15.2 文本的滚动 …… 320
15.3 滚动窗口 …… 323
小结 …… 336
问与答 …… 337
作业 …… 337
测验 …… 337
练习 …… 337

第16天 MFC 控制 …… 338
16.1 静态文本控制 …… 338
16.2 编辑控制 …… 340
CEdit 类 …… 341
与剪贴板有关的编辑函数 …… 342
编辑控制的查询 …… 342
改变编辑控制 …… 345
16.3 下压按钮控制 …… 346
CButton 类 …… 347
处理按钮消息 …… 347
操作按钮 …… 348
命令行计算器 …… 348
小结 …… 365
问与答 …… 365
作业 …… 366
测验 …… 366
练习 …… 376

第17天 分组控制 …… 367
17.1 选中框控制 …… 367
CButton 类和选中框 …… 367
响应选中框消息 …… 368
17.2 圆按钮控制 …… 368

类 CButton 和圆按钮 …… 369
17.3 分组控制 …… 369
类 CButton 与分组框 …… 369
17.4 更新的计算器应用程序 …… 369
关于演示 …… 370
应用程序试验 …… 371
小结 …… 390
问与答 …… 391
作业 …… 391
测验 …… 391
练习 …… 391
第18天 列表框 …… 392
18.1 列表框控制 …… 392
ClistBox 类 …… 392
列表框通知消息的响应 …… 399
列表操作测试程序 …… 399
18.2 处理多项选择列表 …… 409
多项选择列表测试 …… 410
小结 …… 421
问与答 …… 422
作业 …… 422
测验 …… 422
练习 …… 422
第19天 滚动条和组合框 …… 423
19.1 滚动条控制 …… 423
CScrollBar 类 …… 423
响应滚动条通知消息 …… 425
倒计数计时器 …… 426
19.2 组合框控制 …… 434
CComboBox 类 …… 434
响应组合框通知消息 …… 439
组合框用作历史列表框 …… 440
COCA 应用程序第 4 版 …… 440
小结 …… 451
问与答 …… 451
作业 …… 451
测验 …… 451
练习 …… 452
第20天 对话框 …… 453
20.1 构造对话框 …… 453
20.2 执行模式对话框 …… 454
20.3 传送控制数据 …… 459
20.4 模式对话框的数据传送 …… 462
20.5 传送无模式对话框的数据 …… 468
小结 …… 478
问与答 …… 479
作业 …… 479
测验 …… 479
练习 …… 480
第21天 MDI 窗口 …… 481
21.1 MDI 应用程序的特性及其组成 …… 481
21.2 建立一个 MDI 应用程序 …… 482
21.3 CMDIFrameWnd 类 …… 482
21.4 建立 MDI 主窗口 …… 482
21.5 CMDIChildWnd 类 …… 483
21.6 建立 MDI 子窗口 …… 483
21.7 管理 MDI 消息 …… 483
21.8 一个简单的文本查阅器 …… 483
小结 …… 493
问与答 …… 493
作业 …… 494
测验 …… 494
练习 …… 494
第 3 周内容复习 …… 495
附录 A 控制资源描述 …… 502
A.1 建立菜单资源 …… 502
A.2 对话框资源 …… 504
A.3 DIALOG 选项语句 …… 505
STYLE 语句 …… 505
CAPTION 语句 …… 505
MENU 语句 …… 506
CLASS 语句 …… 506
FONT 语句 …… 506
A.4 对话框控制资源 …… 507
A.5 一般控制资源 …… 507
A.6 LTEXT 语句 …… 509
A.7 RTEXT 语句 …… 509
A.8 CTEXT 语句 …… 509
A.9 CHECKBOX 语句 …… 510
A.10 PUSHBUTTON 语句 …… 510
A.11 DEFPUSHBUTTON 语句 …… 510
A.12 LISTBOX 语句 …… 511
A.13 GROUPBOX 语句 …… 511
A.14 RADIOBUTTON 语句 …… 511

A.15 EDITTEXT 语句 …… 512
A.16 COMBOBOX 语句 …… 512
A.17 SCROLLBAR 语句 …… 513
小结 …… 513

附录B 答案 …… 514
第1天 …… 514
第2天 …… 514
第3天 …… 516
第4天 …… 517
第5天 …… 519
第6天 …… 521
第7天 …… 523
第8天 …… 525
第9天 …… 527
第10天 …… 529
第11天 …… 534
第12天 …… 535
第13天 …… 537
第14天 …… 537
第15天 …… 537
第16天 …… 537
第17天 …… 538
第18天 …… 538
第19天 …… 538
第20天 …… 539
第21天 …… 539

词汇表 …… 540

第1周

概要

这是学习编写 Windows 应用程序的第一周，第一天先介绍 Visual C++ 环境——工作平台，其余几天再介绍 C++ 语言本身的基本概念。你将学习到：

- 预定义数据类型
- 定义常量、变量和函数
- C++ 运算符和表达式
- 管理基本输入输出
- 决策
- 编写循环语句
- 声明和使用数组

这样，这周将讲完 C++ 语言的基本组成部分。

第1天

入　门

欢迎你到 C++ 和 Windows 程序设计世界中来! 这次激动人心的探险之旅从今天开始。今天这一课主要是让你熟悉 Visual C++ 工作台。你将学习以下专题:

- C++ 程序基础与历史
- 装载并使用 Visual C++ 工作台
- QuickWin 应用程序
- 键入并运行你的第一个 C++ 程序

1.1　C++ 程序基础

用本书学习 Visual C++ 编程时无需以前有过程序设计经验, 但如果以前编过程序的话, 事情会更好办一些。

和其他语言一样, C++ 也是由声明和描述程序运行时要执行的确切指令的语句组成的。

新　词

面向对象的语言(object-oriented language)
面向对象的语言表示了对象的属性和操作。

另外, C++ 对非面向对象的 C 作了大量的增强。因而, 学习 C++ 的一个额外收获是能使你对 C 非常熟悉。C 已经标准化了, 而 C++ 则不一样, 它正处于标准化过程中。

用 C++ 编程必须了解完成各种任务所用到的支持库, 这些任务包括输入、输出、文本操作、数学运算、文件 I/O(输入/输出)等。在 BASIC 一类的语言中, 对这类操作的支持对程序来说是透明的, 因为程序可自动得到它们。结果, 许多程序部件是作为独立于其他任何程序部件的单独部件而出现的。与之相反的是, 用 C++ 编程要求你更多地注意到不同库之间的依赖关系。这种语言的优越性就在于你能在相似的库, 包括你自己开发的库中进行选择, 因而 C++ 程序是模块化的。C++ 编译器, 包括 Visual C++ , 使用项目文件和程序文件。Visual C++ 工作台使用项目文件来管理程序的建立和更新。

新　词

项目文件(project file)　　程序文件(program file)
项目文件指明库。程序文件建立应用程序。

1.2　装载 Visual C++ 工作台

Visual C++ 工作台(Workbench)是一个包括 C++ 编译器、连接器、调试器以及用

来建立、管理和维护 C+ + 程序的其他工具的可视界面。只要双击 Visual C+ + 图标或在 Windows 的文件管理器(File Manager)中双击 MSVC.EXE 程序即可装载工作台。

Visual C+ + 工作台不使用命令行开关来具体选择其装载和操作的方式,而是使用一菜单选项完成(今天课程的后面将细述)。

1.3 Visual C+ + 工作台概观

Visual C+ + 工作台是供开发 C 和 C+ + 应用程序的环境。工作台是一个有着多种用途的工具,用它可以更方便地开发 MS-DOS、Windows 的应用程序和库。在整个这本书中,你将用工作台来学习用 C+ + 进行程序设计并编写 Windows 应用程序。

Visual C+ + 工作台包括菜单选项、工具栏、状态栏和显示区。内置的编辑器用此区域显示和编辑在 MDI(多文档界面)窗口中的文件。

工作台的一个极有用的特点是广泛的联机帮助功能。可以用 Help 菜单项询问各种有关工作台、C+ + 语言以及程序开发的其他方面的问题。

Visual C+ + 工作台的菜单共有十个菜单项。不必细究所有这些选项的确切含义,这里只是概述一下。每个菜单项都包含有一族完成某个任务或设置某一选项的选择项。以下各节介绍 Visual C+ + 工作台的菜单项。

1.4 File 菜单项

File 菜单项主要管理装入和保存文件、打印以及退出工作台等工作。此菜单项包括以下各个小节中的选择项。

New 选择项

New 选择项建立一个新的空白的 MDI 编辑窗口。每一个 MDI 窗口有一括在尖括号内的窗口号和一标题(文件名)。第一个新的 MDI 窗口标题是 UNTITLED.1,第二个新的 MDI 窗口标题则为 UNTITLED.2,依此类推。

Open 选择项

Open 选择项打开一现有文件。该选择项弹出一 Open File 对话框,如图 1.1 所示,供你选择文件。该对话框允许你从现有目录和驱动器上选择一文件。另外,对话框有一通配文件名文本框,可供你在选中打开单个的文件之前选择多个文件察看。对话框还包括一 Read Only 选择框。此控制允许你读文件并可防止偶然改动。

Close 选择项

Close 选择项关闭当前编辑的 MDI 窗口。如果窗口中的内容未被保存或更新,则 Close 选择项显示一消息对话框。该对话框询问你是否在关闭窗口前保存其内容。

图 1.1　Open File 对话框

Save 选择项

Save 选择项更新与当前编辑窗口相关联的文件。如果在新窗口中选此选项，则工作台会产生 Save As 对话框。

Save As 选择项

用 Save As 选择项可以保存新窗口或将当前编辑窗口的内容写到另一文件中去。此选择项产生 Save As 对话框，可以指定文件名以及目标驱动器和目录(见图 1.2)。

图 1.2　Save As 对话框

Save All 选择项

Save All 选择项可快速保存工作台中的所有窗口的内容。对于新窗口则产生 Save As 对话框来保存它。

Print 选择项

用 Print 选择项可以打印当前编辑窗口的部分或全部内容。此选择项产生 Print 对话框,可以选择打印机、全部或部分打印所编辑的文本,以及调用打印机设置程序。Print 对话框产生其他对话框进行打印机设置。

Page Setup 选择项

Page Setup 选择项产生的对话框供你指定页眉(header)、页脚(footer)以及打印文档的 4 个边缘。

Exit 选择项

用 Exit 选择项可退出 Visual C+ + 工作台。如果有未保存的编辑窗口,则工作平台会给你一个机会,以便在退出前将这些窗口的内容保存起来。

Most Recent Files 选择项

Visual C+ + 工作台会记住你工作过的最近几个文件。工作台在 File 菜单项的 Exit 选择项下面显示这些文件的名字。

1.5　Edit 菜单项

Edit 菜单项中包含进行以下 4 种操作的选择项:取消更改(undoing changes),剪切和粘贴操作,查找/替换操作,以及设置/取消只读方式。

Undo 选择项

用 Undo 选择项可取消前一次的编辑操作。可取消的编辑操作的数量取决于 Undo 缓冲区(用来跟踪和存储文本的更改部分的内存区)的大小。缓冲区的缺省大小是 4K,实际可设为 0 到 64K 之间的任意大小值。用 Options 菜单项中的 Edit 选择项可以改变 Undo 缓冲区的大小。

Redo 选择项

Redo 选择项取消由 Undo 选择项完成的动作。如果使用 Undo 选择项的次数过多,那么 Redo 选择项可以帮助恢复部分编辑文本。

Cut 选择项

Cut 选择项删除选中的文本并将其拷贝到剪贴板中(覆盖剪贴板中原来的内容)。可

以用 Paste 选择项重新将被删除的文本插入。

Copy 选择项

Copy 选择项将选中的文本拷贝到剪贴板中, 覆盖掉剪贴板中原来的内容。接着再选择 Paste 选择项可以插入被拷贝的文本。

Paste 选择项

Paste 选择项将剪贴板中的文本插入到当前光标位置处。可以用 Cut 和 Paste 选择项将文本从一处移到另一处。可以用 Copy 和 Paste 选择项在一个或多个窗口中复制文本。

Delete 选择项

Delete 选择项删除选中的文本或光标位置处的字符(如果没有文本被选中)。被删除的文本或字符并不复制到剪贴板中。

Find 选择项

用 Find 选择项可在当前编辑窗口中搜索文本。该选择项产生 Find 对话框, 如图 1.3 所示。此对话框允许你指定以下搜索选项:

图 1.3　Find 对话框

- 搜索文本(对话框用一组合框控制供你指定以前搜索过的文本)
- 搜索整个单词(对话框提供一选择框以指定是否搜索整个单词)
- 区分大小写搜索(对话框提供一选择框以指定是否区分大小写搜索)
- 使用具有更高级的文本模式的正则表达式来搜索文本
- 搜索方向(对话框提供圆按钮指定搜索方向)

Find 对话框允许你一次搜索一个匹配实例, 或者在包含有匹配文本的所有行上设置上标记(bookmark)。

Replace 选择项

Replace 选择项在当前编辑文本中替换文本。该选择项产生一 Replace 对话框, 如图 1. 4 所示。该对话框允许你指定如下搜索选项:

- 搜索文本(对话框用一组合框控制供你指定以前搜索过的文本)
- 替换文本(对话框用一组合框控制供你指定以前替换过的文本)
- 搜索整个单词(对话框提供一选择框以指定是否搜索整个单词)
- 区分大小写搜索(对话框提供一选择框以指定是否区分大小写搜索)
- 使用具有更高级的文本模式的正则表达式搜索文本
- 搜索方向(对话框提供圆按钮指定搜索方向)

图 1. 4 Replace 对话框

Replace 对话框用替换文本在被搜索文本的所有出现之处进行替换。该对话框还包含一 Find Next 命令按钮可供你搜索下一个。

Find Matching Brace 选择项

用 Find Matching Brace 选择项可查找匹配的花括号——{和}。C+ + 用花括号定义语句块。

Read Only 选择项

Read Only 选择项设置/取消窗口中文本的只读方式。当设置只读方式时, 工作平台将一选中标记置于此选择项的左边。

1.6　View 菜单项

用 View 菜单项可以迅速察看各种不同的信息。此菜单包括察看一特定行、错误、标记、工具栏和状态栏的选择项。

Line 选择项

Line 选择项产生一对话框提示你输入所要察看的行的行号。用此选择项可以精确地跳转到一特定行。当在一很大的文件中需要跳转很远的距离时此选择项尤其有用。

Mixed Source/ Asm 选择项

Mixed Source/ Asm 选择项显示 C+ + 语句与汇编语言的混合代码。此选择项是供高级程序员使用的。

Next Error 选择项

用 Next Error 选择项可以看下一个错误行。如果非法行所在的文件不在当前的 MDI 窗口中,那么该选择项将装入该文件。

Previous Error 选择项

用 Previous Error 选择项可以看前一个错误行。如果非法行所在的文件不在当前的 MDI 窗口中,那么该选择项将装入该文件。

Toggle Bookmark 选择项

Toggle Bookmark 选择项设置/取消对光标所在行的强调显示。当该选择项强调显示一行时,它在该行设置一标记。该选择项清除该行的强调显示时,该行的标记将不复存在。

Next Bookmark 选择项

用 Next Bookmark 选择项可访问下一个带标记的行。可以用 Toggle Bookmark 选择项或用 Edit 菜单项中的 Find 选择项建立这样的行。

Previous Bookmark 选择项

用 Previous Bookmark 选择项可访问前一个带标记的行。

Clear All Bookmarks 选择项

Clear All Bookmarks 选择项去掉所有的标记。在用 Edit 菜单项中的 Find 选择项来标记与特定文本匹配的行之前,必须选用此项。

Toolbar 选择项

Toolbar 选择项设置/取消工作台的工具栏。当选中此选项时,工具栏是可见的。

Status Bar 选择项

Status Bar 选择项设置/取消工作台的状态栏。当选中此选择项时,状态栏是可见的。

Syntax Coloring 选择项

可以用 Syntax Coloring(语法着色选择项为 C 或 C+ + 程序选择彩色的语法表示。该选择项显示一级联菜单以选择 C、C+ + 或不选。后者关闭彩色语法表示。

1.7　Project 菜单项

Project 菜单项用来管理应用程序的建立和更新。可以用 Project 菜单中的各选择项来编译和运行 C+ + 程序。

AppWizard 选择项

AppWizard 选择项可以帮助建立使用 Microsoft 基本类(MFC)库的 Windows 应用程序。此选择项打开 AppWizard 对话框供你生成与 ClassWizard(另一用来建立类的强大工具)一起使用的 MFC 项目文件族。

New 选择项

New 选择项可以帮助建立新的项目文件。此选择项弹出一 New Project 对话框,你可以指定项目名、程序类型(MS-DOS 或 Windows),以及加入建立程序时要调用的文件。

Open 选择项

Open 选择项打开一现有的项目并装入所需文件。用此选择项可以在一已有项目中完成、更新或运行程序。此选择项自动关闭当前打开的项目。工作台一次只处理一个项目。

Edit 选择项

Edit 选择项打开一现有的项目文件并使得你能向所需的文件列表中加入或从中删除文件。不能用此选择项改变项目文件所生成的文件的类型。要做到这一点,请在Options 菜单中选用 Project 选择项。

Close 选择项

Close 选择项显式地关闭一个项目。如果退出工作台时没有关闭当前项目,那么下次装载工作台时将自动装入项目文件。

Compile File 选择项

Compile File 选择项编译在当前活动窗口中的程序。如果需要的话,将保存当前窗口的内容。工作台将显示一输出窗口,在该窗口中显示编译状态和编译器所产生的错误报告。

新　词

源文件(sourcefile)
在 C++ 中,可以用几个模块生成一个程序。每一个模块都存储在一个源文件中。编译器根据每个源文件生成.OBJ 目标文件,连接程序将不同的.OBJ 文件和运行库文件组装起来就生成了.EXE 程序文件。

Build 选择项

Build 选择项通过编译(或重编译)源文件生成.EXE 程序文件。该选择项仅对以下文件起作用:

- 未编译过的
- 先前编译过,但其相应的源代码已作修改

项目文件中存储着描述某一文件对另一些文件的依赖关系的信息。此信息使得 Build 选择项可以确定文件之间的依赖关系的层次结构。

Rebuild All 选择项

Rebuild All 选择项通过系统地编译其所有模块来生成.EXE 程序文件。

Stop Build 选择项

Stop Build 选择项可立即停止应用程序的生成。

Execute 选择项

Execute 选择项运行项目中的程序。如果需要,此选择项重新编译和链接源文件模块。在本书中,你将用此选项运行程序。

Scan Dependencies 选择项

Scan Dependencies 选择项更新当前所编辑文件的依赖文件的列表。

Scan All Dependencies 选择项

Scan All Dependencies 选择项更新整个项目中所有依赖文件的列表。

Load Workspace 选择项

用 Load Workspace 选择项可以装入三个工作间的任一个(Edit, Debug 和 Custom)。工作间中包括窗口的安排。

Save Workspace 选择项

用Save Workspace 选择项可以将当前工作间存到三个工作间(Edit, Debug 和 Custom)中的任一个中去。

Most Recent Projects 选择项

Project 菜单中包括最多可达 4 个的最近工作过的.MAK 项目文件。可以在此选择项

中快速调出一个项目。

1.8　Browse 菜单

Browse 菜单包含变化复杂的各种选项。有些选择项可以完成简单的任务,另一些则要等到后来才能发挥作用。

Go to Definition 选择项

Go to Definition 选择项跳转到定义名字的第一个地方。如果该定义不在工作间的任何窗口中,那么该选择项将装入包含有定义的文件。

Go to Reference 选择项

Go to Reference 选择项跳转到引用名字的第一个地方。如果该引用不在工作间的任何窗口中,那么该选择项将装入包含有引用的文件。

Next 选择项

Next 选择项跳转到名字的下一次定义或引用处。此选择项的动作取决于上两个选择项中的哪一个最近被调用过。

Previous 选择项

Previous 选择项跳转到名字的前一次定义或引用处。此选择项的动作取决于 Go to Definition 或 Go to Reference 这两个选择项中的哪一个最近被调用过。

Pop Context 选择项

Pop Context 选择项跳转到上一次调用 Go to Definition 或 Go to Reference 选择项之前用过的名字处。

Open 选择项

Open 选择项打开 Browse 窗口以供你在程序的各个模块中浏览。在要建立模块化程序时,此高级选择项很有作用。

ClassWizard 选择项

ClassWizard 选择项打开 ClassWizard 对话框。此对话框可帮你编写用 MFC 库的 Windows 程序。

1.9　Debug 菜单

用 Debug 菜单中的选择项可以监控程序的执行,从而对运行时的错误进行追踪。调试方法是在特定的语句处设置断点。用 Debug 菜单中的选择项还可以单步执行程序并观

察变量的值。

新　词

断点(breakpoint)
断点使程序临时停止,以便对各个模块进行检查。

1.10　Tools 菜单

Tools 菜单是可以由用户自己来定义的灵活菜单,其选择项是可以从工作台中调用的实用程序。缺省的选择项是 App Studio, CodeView for Windows 以及 CodeView for MS-DOS。工作台 Options 菜单中的 Tools 选择项可以修改 Tools 菜单中的选择项。

1.11　Options 菜单

用 Options 菜单可以详细调整编译器、连接程序和 Visual C+ + 工作台的其他操作。

Project 选择项

Project 选择项供用户调整编译器、连接程序、资源、生成方式和项目类型。此选择项产生 Project Options 对话框,后者再产生其他对话框以调整编译器、连接程序和资源的操作。

Debug 选择项

用 Debug 选择项可以调整程序的调试方法。

Directories 选择项

Directories 选择项产生 Directories 对话框,它设置 include 文件、库、可执行文件、求助文件和 MFC 库源文件的目录。

Editor 选择项

Editor 选择项弹出 Editor 对话框,它设置的选项包括制表站设定、滚动条的使用、在项目生成(build)前保存文件、在保存文件前进行提示、Undo 缓冲区的大小等。

Workspace 选择项

Workspace 选择项打开 Workspace 对话框并让你设置三个工作间选项。

Tools 选择项

Tools 选择项打开 Tools 对话框,用它可以向 Tools 菜单中增加或从中删除工具。

Color 选择项

Color 选择项弹出 Colors 对话框,用它可以改变用于语法着色的颜色。

Fonts 选择项

Fonts 选择项产生 Fonts 对话框,用它可以为编辑窗口选择字体。

1.12 Window 菜单项

Window 菜单中包含一组可用来将窗口在工作间中平铺、级联、关闭和进行复制的选择项。另外,用 Window 菜单可显示观察程序变量和 CPU 寄存器的专门窗口。

1.13 Help 菜单

Help 菜单提供了多方位的省时的联机帮助。Help 菜单中的选择项所提供的内容包括工作台本身、生成工具、C/C+ + 语言、MFC 库、Windows 3.1 SDK(软件开发工具)以及何处获取技术信息。

1.14 QuickWin 应用程序

用 Visual C+ + 工作台可以建立一种特殊的程序,称为 QuickWin 应用程序。此应用程序介于 MS-DOS 程序和 Windows 程序之间。第 1 天到第 12 天的程序都是 QuickWin 应用程序,这使得你可以用像 DOS 一样的界面和输入/输出来集中学习 C+ + 。QuickWin 窗口是用于 C+ + 程序(按 QuickWin 应用程序编译)的标准输入和输出。QuickWin 窗口的菜单带少数菜单项和几个选择项。

若要建立一 QuickWin 应用程序,按以下步骤操作:

1. 装入 Visual C+ + 工作台。
2. 选择 File 菜单中的 New 选择项。
3. 在新窗口中键入 C+ + 程序。
4. 选择 File 菜单中的 Save As 选择项,用. CPP 扩展名保存 C+ + 程序。Save As 对话框允许你选择驱动器与目录以存储新程序。
5. 用 File 菜单中的 Close 选择项,关闭当前编辑窗口。
6. 选择 Project 菜单中的 New 选择项弹出 Project 对话框。
7. 键入项目名(用与. CPP 文件同样的名字)并从 Project Type 列表框中选择 QuickWin 应用程序项。做完后单击 OK 按钮。
8. 工作台显示 Edit 对话框供你选择组成项目的文件。该对话框中包括用来选择另一驱动器与另一目录的控制。用这些控制可以决定刚刚键入的程序存于何处。若要将文件加入到项目文件列表中,只要双击该文件或者选中它并单击 Add 按钮。若要从项目文件列表中去掉文件,可选中该文件并单击 Delete 按钮。当选完文件后,单击 Close 按钮。
9. 选取 Project 菜单中的 Build 选择项。这一步将编译并链接你的程序。
10. 选择 Project 菜单中的 Execute 选择项运行该程序。输出将显示在与项目文件同

名的 QuickWin 窗口中。

若要察看项目文件,可单击工具栏左边的图标。此操作使工作台显示组成项目的所有文件。双击想要察看和/或编辑的文件。

> **注　意**　本书中的程序都应在目录\MSVC\VC21DAY 下。请在\MSVC 目录下建立子目录 VC21DAY。

1.15　你的第一个 C++ 程序

本书所提供的第一个 C++ 程序显示一行问候信息。这个简单的程序可使你了解 C++ 程序的基本组成部分。

清单 1.1 中列出了程序 HELLO.CPP 的带行号的源代码。输入程序的时候不要将源代码也键入。这个程序很简单,只是显示串"Hello Programmer!"。按以下步骤操作以建立和运行第一个 C++ 程序:

1. 如果 Visual C++ 工作台还未装载,则将其装入。
2. 选取 File 菜单中的 New 选项。
3. 在新窗口中键入清单 1.1 中列出的程序。
4. 选取 File 菜单中的 Save As 选择项。将 C++ 程序保存为\MSVC\VC21DAY 目录下的 HELLO.CPP。
5. 用 File 菜单中的 Close 选项关闭当前编辑窗口。
6. 选取 Project 菜单中的 New 选项。此步骤弹出如图 1.5 所示的对话框。
7. 键入 hello 作为项目的名字并从 Project Type 列表框中选择 QuickWin 应用程序(.EXE)项。单击 OK 按钮。
8. 工作台显示 Edit 对话框,如图 1.6 所示。选中目录\MSVC\VC21DAY 下的 HELLO.CPP 文件并双击该文件。
9. 选取 Project 菜单中的 Build 选项以编译和链接程序 HELLO.CPP。
10. 选择 Project 菜单中的 Execute 选项以运行程序 HELLO.EXE。

键　入　　清单 1.1　程序 HELLO.CPP 的源代码

```
1: // a trival C++ program that says hello
2:
3: # include < iostream.h>
4:
5: main()
6: {
7:     cout < < "Hello Programmer!"
8:     return 0
9: }
```

图 1.5　New Project 对话框

图 1.6　Edit 对话框

程序的输出显示在图 1.7 中。注意输出窗口的状态栏有一词“Finished”，它表示程序已经终止。

图 1.7　程序 HELLO.EXE 的输出

分　析

仔细阅读此 C++ 程序的简短代码，注意以下特点：

· C++ 使用“//”字符作为一直延伸到行尾的注释。C++ 也支持以“/*”字符开始、以“*/”结束的 C 风格的注释。第 1 行中就有简单描述程序的注释。

新　词

注释(comment)
注释是放在程序中的用来解释或澄清程序的某一部分的评述。编译器会忽略掉注释。

· C++ 程序中没有声明程序结束的保留关键字。实际上，C++ 使用一非常简单的模式来组织程序。此模式支持两级代码层次：全局的以及具有单一层次的函数。另外从第 5 行开始的函数 main 起着非常特殊的作用，因为程序运行时是从此函数开始执行的。因此，C++ 程序中只能有一个 main 函数。可以将 main 放在程序中的任何地方。

· C++ 中串和字符分别用双引号和单引号括起来。因而，'A' 是一字符而"A"则是一单字符的串。不可将 C++ 中单字符的串和字符混为一谈。

新　词

空串(empty string)
串中可含任意数目的字符，甚至没有字符——空串。

· C++ 用“{”和“}”字符定义块(分别见例子中的第 6 行和第 9 行)。

· C++ 程序中的每一条语句都必须以分号(;)结尾。

· C++ 中有# include 编译指令。第 3 行就是一个例子，它指示 Visual C++ 编译

器将 IOSTREAM. H 头文件包括进去。C+ + 扩展了 C 中已有的流的概念，IOSTREAM. H 提供的操作可支持基本流输入和输出。C+ + 语言不包括内置的 I/O 例程，而是依赖于专用于不同类型 I/O 的库。

新 词

编译指令(compiler directive)　**头文件**(header file)　**流**(stream)

编译指令是专用于编译器的特殊指令。头文件中包括常量的声明、数据类型、变量、函数的提前声明。流是从计算机中某部分到另一部分的数据流序列。

- C+ + 程序将串"Hello Programmer! "输出到标准输出流 cout，也就是 QuickWin 窗口中。另外，程序还使用提取操作符"< < "将输出串发送到输出流。
- main 函数必须返回一个反映 C+ + 程序的出错状态的值。返回 0 值则通知操作系统程序正常终止。

1.16 退出工作台

若要退出工作台，可关闭当前项目(用 Project 菜单中的 Close 选项)并选择 File 菜单中的 Exit 选项。

小　结

今天这一课介绍了 Visual C+ + 工作台并给你提供了第一个 C+ + 程序。基本内容可概括如下:

- C+ + 程序是模块化的且依赖于标准的或用户定义的库。
- 装载 Visual C+ + 工作台有两种方法: 单击 Visual C+ + 图标或在文件管理器(或任何类似的实用程序)中双击 MSVC. EXE 文件。
- Visual C+ + 工作台是可供开发、维护和调试应用于 MS-DOS 和 Windows 的 C 和 C+ + 程序的全方位环境。
- 用 File 菜单可以管理文件、打印文本以及退出工作台。
- 用 Edit 菜单项可以取消对文本所做的更改、管理文本(剪切、粘贴和删除)、搜索文本以及替换文本。
- 用 View 菜单可以查看编译错误、管理标记(bookmark)、切换状态栏、切换工具栏。
- 用 Project 菜单可以通过编译源文件、生成(build)项目、运行程序、管理工作间以及管理依赖文件来管理程序。
- 用 Browse 菜单可以查找名字的定义和引用。
- 用 Debug 菜单提供的选项可以管理源文件中的断点、单步执行程序并观察变量。
- 用 Tools 菜单可以调出辅助开发、管理和维护程序的实用程序。
- Options 菜单微调项目、目录、编辑器、语法着色、字体、工具、工作间和调试器。
- 用 Window 菜单可以管理窗口以及打开特殊窗口。

- Help 菜单提供全方位的各种主题的联机帮助，例如工作台、C/C++ 语言以及 MFC 库。
- QuickWin 应用程序是提供类似于标准输入和输出设备的特殊窗口的 Windows 应用程序。QuickWin 应用程序允许你编写 DOS 一类的程序。
- 第一个 C++ 程序是一简单的问候程序，它示意了 C++ 程序的基本组成部分。这些部分包括注释、# include 指令和 main 函数。
- 用 File 菜单中的 Exit 选项可退出工作台。

问　与　答

问：C++ 中使用行号吗？

答：不用。在清单中使用行号仅仅为了引述方便。

问：工作台的编辑器是否对键盘输入进行监控？

答：是的。事实上，当键入一个 C++ 关键字时，工作台迅速给此关键字着色。

问：上面的程序中，如果忘了键入第二个引号会发生什么？

答：编译器会告诉你程序中有错误。必须将第二个引号加上并进行项目生成。

问：如何在当前编辑窗口中删除文本？

答：用 Edit 菜单中的 Replace 选项并指定替换串为空串。

作　　业

作业中所提供的测验题帮助你巩固所学内容，而练习则就使用所学内容提供一些经验。要弄懂测验和练习的解答后再继续下一天的课程。附录 B 的“答案”中提供了解答。

测　验

1. 以下程序的输出是什么？

```
1: //quiz program # 1
2:
3: # include < iostream.h>
4:
5: main()
6: {
7: cout < < "C++ in 21 Days?";
8: return 0;
9: }
```

2. 以下程序的输出是什么？

```
1: // quiz program # 2
2:
3: # include < iostream.h>
4:
```

```
5: main()
6: {
7: // cout < < "C+ + in 21 Days?"
8: return 0;
9: }
```

3. **下面的程序错在哪儿?**

```
1: // quiz program # 3
2:
3: # include < iostream. h>
4:
5: main()
6: {
7: cout < < "C+ + in 21 Days?"
8: return 0;
9: }
```

练　习

编写程序显示一行“I am a C+ + Programmer.**”。**

第2天

C++ 程序的组成

在第 1 天中我们讲述了 Visual C++ 工作平台和一个简单的 C++ 程序。今天的重点是 C++ 程序的基本组成部分，包括数据类型、变量、常量和函数。你将学到如下内容：

- Visual C++ 中的预定义数据类型
- 在 Visual C++ 中命名项目
- # include 指令
- 声明变量
- 声明常量
- 声明函数及定义其原型
- 函数中的局部变量
- 函数中的静态变量
- 嵌入(inline)函数
- 退出函数
- 缺省变元
- 函数重载

2.1 Visual C++ 中的预定义数据类型

Visual C++ 中提供 int，char，float，double 以及 void 数据类型分别表示整数、字符、单精度浮点数、双精度浮点数以及无值数。C++ 将 void 类型用于函数的返回值表示该函数作为过程使用。

C++ 支持数据类型修饰词，使得数据类型更加灵活。类型修饰词是：signed，unsigned，short 和 long。表 2.1 显示了 C++ 中的预定义数据类型(包括类型修饰词)以及它们的大小和值域。注意 int 和 unsigned int 跟系统有关。表中显示了这些数据类型的 16 位值。

新 词

数据类型修饰词(type modifier)
数据类型修饰词改变值的精度和范围。

新 词

十六进制数(hexadecimal number)
C++ 中支持十六进制数。这类数以字符 0x 开始，后面跟以十六进制值。例如，0xff 就是与十进制数 255 相等的十六进制数。

表 2.1　C+ + 中的预定义数据类型

数据类型	字节大小	值　域	举　例
char	1	- 128 到 127	'A', '©'
signed char	1	- 128 到 127	23
unsigned char	1	0 到 255	200, 0x1a
int	2	与系统有关, 16 位对应 - 32768 到 32767	3000
unsigned int	2	与系统有关, 16 位对应 0 到 65535	0xffff, 65535
short int	2	- 32768 到 32767	100
unsigned short int	2	0 到 65535	0xff, 40000
long int	4	- 2147483648 到 2147483647	0xfffff - 123456
unsigned long int	4	0 到 4294967295	123456
float	4	3.4E - 38 到 3.4E + 38 以及 - 3.4E - 38 到 - 3.4E + 38	2.35 - 52.354 1.3e + 10
double	8	1.7E - 308 到 1.7E + 308 以及 - 1.7E - 308 到 - 1.7E + 308	12.354 - 2.5e + 100 - 78.32544
long double	10	1.2E - 4932 到 1.2E + 4932 以及 - 1.2E - 4932 到 - 1.2E + 4932	8.5e - 3000

2.2　在 Visual C+ + 中命名标识符

注意 Visual C+ + 中对标识符有以下规定:

1. 第 1 个字符必须是字母或下划线(_)。
2. 后续字符可以是字母、数字或下划线。
3. 标识符的最大长度是 247 个字符。
4. 在 C+ + 中标识符是大小写相关的。因而 rate, RATE 和 Rate 指的是三个不同的标识符。
5. 标识符不能使用保留关键字,例如 int, double 或 static(这里只列举了少数几个)。

合法标识符举例:

```
X
x
aString
DAYS_IN_WEEK
BinNumber0
bin_number_0
bin0Number2
```

- length

要用描述性的名字且长度适中。
不要使用名字太长或太短的标识符。太短的名字可读性差,名字太长打字容易出错。

2.3　# include 指令

回忆一下,指令(directive)是对 C 和 C+ + 编译器的特殊指示。指令以“# ”字符开头,后跟以一指令名。指令通常置于一行的第一列,前面只能有空格或制表站(tab)字符。第 1 天中的 C+ + 程序就包含有# include 指令。这条指令让编译器包含进某个文件的文本就好像程序员键入该文本一样。用# include 指令这种做法比将文本从一个文件中剪切下来并粘贴到另一文件中的方法更好些。回头看看第 1 天,程序使用了# include 指令包含头文件。

语　法

include 指令

include 指令的一般语法是:

```
# include < filename>
```

示　例:

```
# include < iostream.h>
# include "string.h"
```

filename 表示包含文件的名字。这两种形式的区别在于# include 指令搜索包含文件的方法不同。第一种形式仅在专用于包含文件的特殊目录中搜索文件,第二种形式将搜索范围扩大到包括当前目录在内。

2.4　声 明 变 量

声明变量时要指明变量的数据类型和变量名。“变量”一词表示你可以修改这些数据存储器中的数据。

新　词

变量(variable)

变量是用来存储和检索信息的标识符。可以把变量看作一个带标号的数据存储器。

语　法

声明变量

声明变量的一般方法是:

```
type variableName;
type variableName = intialValue;
type var1 [= initVal1], var2 [= initVal2], ...;
```

示　例:

```
int j;
double z = 32.314;
long fileSize, diskSize, totalFileSize = 0;
```

C+ + 中允许在一条声明语句中声明多个变量。例如:

```
int j, i = 2, k = 3;
double x = 3.12;
double y = 2 * x, z = 4.5, a = 45.7;
```

初始化值可包含其他先前已定义的变量。

要尽量不用全局变量。

不要在同一程序单元内声明仅靠大小写区别名字的两个变量。

我们来看一个使用变量的简单程序。清单 2.1 中列出了程序 VAR1.CPP 的源代码。程序声明了 4 个变量,其中两个是在声明过程中初始化的。该程序给未初始化的变量赋值并将 4 个变量的内容显示出来。建立项目 VAR1 并包含 VAR1.CPP 文件。生成并运行 VAR1.MAK 项目。

键　入　　**清单 2.1　程序 VAR1.CPP 的源代码**

```
1: //C+ + program that illustrate simple variables
2:
3: # include < iostream.h>
4:
5: main()
6: {
7:   int i, j = 2;
8:   double x, y = 355.0 / 113;
9:
10:   i = 3 * j;
11:   cout < < "i \ " < < i < < "\n"
12:        < < "j = " < < j < < "\n";
13:
14:   x = 2 * y;
15:   x = x * x;
16:   cout < < "y = " < < y < < "\n"
17:        < < "x = " < < x < < "\n";
18:   return 0;
19:
20:   }
```

以下是清单 2.1 中程序的一个运行范例:

输　出

```
i =  6
j =  2
y =  3.14159
x =  39.4784
```

分　析

本程序在第 3 行用# include 指令包含流 I/O 头文件 IOSTREAM.H。main 函数出现在第 5 行。该函数在第 7 行声明了整型变量 i, j; 第 8 行则声明了双精度型变量 x 和 y。变量 j 和 y 在声明时被初始化。第 10 行的语句将变量 j(等于 2) 的值乘以 3 并将结果存在变量 x 中。第 11 行和第 12 行的流输出语句显示变量 i 和 j 的值, 该语句中还包含标记输出的串。

第 14 行的语句将变量 y 的值乘以 2 并将其存于变量 x 中。第 15 行的语句求变量 x 的平方值并将结果回送到 x 中, 该语句在等号两边都使用变量 x。第 16 和 17 行的流输出语句显示了变量 x 和 y 的值。第 18 行中的语句返回 0 作为 main 函数的结果。

2.5　声 明 常 量

许多语言, 如 BASIC(最近的实现版本), Modula-2, Ada, C, Pascal 以及 C++ 都支持常量。毋庸置疑, 常量用更具描述性的标识符来代替数值常数, 这样增强了程序的可读性。更有一点, 使用常量时只要在一个地方修改参数值就可以改变程序中所有参数值。这一特性自然会带来极大的方便, 并且不易导致在文本编辑器中用一数替换另一数时可能会出现的错误。

新　词

常量(constant)

常量是与固定值相关联的标识符。C++ 中提供了两种风格的常量: 基于宏的和正规的。基于宏的常量是从 C 继承下来的并使用# define 编译指令来定义。

语　法

define 指令

define 指令的一般语法是:

```
# define constantName constantValue
```

define 指令让编译器调用预处理程序进行文本替换, 用其数值替换基于宏的常量。这一文本替换步骤发生在编译器处理源文件中的语句之前。再往后, 编译器再也不会看见基于宏的常量本身——只看见它们展开后的内容。

示　例:

```
# define ASCII_A 65
# define DAYS_IN_WEEK 7
```

C++ 中常量的第二种类型是正规常量。

语　法

正规常量

正规常量的一般语法是:

```
const dataType constantName = constantValue;
```

dataType 项是一可选项,它指定常量值的数据类型。如果省略数据类型,则 C++ 编译器假设其为 int 类型。

示　例:

```
const unsigned char ASCII_A = 65;
const DAYS_IN_WEEK = 7;
const char FIRST_DISK_DRIVE = 'A';
```

要用大写字母作常量命名。这种命名风格使你可以迅速确定一个标识符是不是常量。

不要指望别人阅读你的代码时会明白你所嵌入的数的含义。使用常量声明能增强程序的可读性。

使用基于宏的常量

先看一个使用基于宏的常量的例子。清单 2.2 中列出了 CONST1.CPP 程序的源代码。该程序提示你输入从子夜开始的小时、分钟以及秒钟数,然后计算出从子夜开始的总的秒数。建立项目 CONST1 并包含 CONST1.CPP 文件。生成并运行 CONST1.MAK 项目。

键　入　　**清单 2.2　程序 CONST1.CPP 的源代码**

```
1: // C++ program that illustrates constants
2:
3: #include <iostream.h>
4:
5: #define SEC_IN_MIN 60
6: #define MIN_IN_HOUR 60
7:
8: main()
9: {
10:   long hours, minutes, seconds;
11:   long totalSec;
12:
13:   cout << "Enter hours: ";
14:   cin >> hours;
15:   cout << "Enter minutes: ";
16:   cin >> minutes;
17:   cout << "Enter seconds: ";
18:   cin >> seconds;
19:
20:   totalSec = ((hours * MIN_IN_HOUR + minutes) *
```

```
21:                    SEC_IN_MIN) +  seconds;
22:
23:   cout << "\n\n" << totalSec << " seconds since midnight";
24:   return 0
25:   }
```

以下是清单 2.2 中程序的一个运行范例:

输　出

```
Enter hours: 10
Enter minutes: 0
Enter seconds: 0

36000 seconds since midnight
```

分　析

本程序在第 3 行中用# include 指令包含头文件 IOSTREAM.H。第 5 和第 6 行中的 # define 指令声明基于宏的常量 SEC_IN_MIN 和 MIN_IN_HOUR。这两个常量的值都是 60, 但含义并不同。从第 8 行开始的 main 函数声明了 4 个 long 类型的变量 hours, minutes, seconds 和 totals。

main 函数中用了好几对语句以输出提示信息并接收输入。第 13 行中的流输出语句提示输入小时数。第 14 行是一个流输入语句。标识符 cin 是标准输入流的名字, 它使用插入操作符">>"从键盘读取数据并将其存在变量 hours 中。第 15 到第 18 行的输入和输出语句完成类似的操作, 即提示输入并获取键盘输入。

第 20 行中的语句计算从子夜开始的总秒数并将结果存于变量 totalSec 中。该语句使用了基于宏的常量 MIN_IN_HOUR 和 SEC_IN_MIN。显然, 用这些常量比用 60 代替这两个常量增强了程序的可读性。第 23 行的流输出语句显示了从子夜开始的总秒数(存于变量 totalSec 中), 后面跟以描述文字以使输出更加清晰。

使用正规常量

现在来看看上述程序的一个新版本, 它使用了正规的 C++ 常量。清单 2.3 中列出了程序 CONST2.CPP 的源代码。此程序工作原理与 CONST1.CPP 一样。建立项目 CONST2 并包含 CONST2.CPP 文件。生成并运行 CONST2.MAK 项目。

键　入　　**清单 2.3　程序 CONST2.CPP 的源代码**

```
1: // C++ program that illustrates constants
2:
3: # include <iostream.h>
4:
5: const SEC_IN_MIN =  60;
```

```
6:
7: main()
8: {
9: const MIN_IN_HOUR = 60; // local constant
10:
11: long hours, minutes, seconds;
12: long totalSec;
13:
14: cout << "Enter hours: ";
15: cin >> hours;
16: cout << "Enter minutes: ";
17: cin >> minutes;
18: cout << "Enter seconds: ";
19: cin >> seconds;
20:
21: totalSec = ((hours * MIN_IN_HOUR + minutes) *
22:               SEC_IN_MIN) + seconds;
23:
24: cout << "\n\n" << totalSec << " seconds since midnight";
25: return 0
26: }
```

以下是清单 2.3 中程序的一个运行范例:

输　出

```
Enter hours: 1
Enter minutes: 10
Enter seconds: 20

4220 seconds since midnight
```

分　析

清单 2.2 中的程序与 2.3 中的很相似。它们的区别在于声明常量的方式不同,清单 2.3 中用的是正规的 C+ + 常量语法声明常量。另外,在 main 函数的外面第 5 行中声明了常量 SEC_IN_MIN。这种声明方法使得常量成为全局性的。也就是说,如果程序中有另一函数,那么它也可以使用常量 SEC_IN_MIN。与之相反,常量 MIN_IN_HOUR 是在 main 函数内部定义的。因而,常量 MIN_IN_HOUR 局限于函数 main 内。

2.6　声明函数及定义其原型

许多程序设计语言都使用函数和过程。C+ + 不支持正规的过程,所有的 C+ + 例程都是函数。

新　词

函数(function)

函数是从概念上扩展 C+ + 语言以适应专门的用户程序的主要构件。

语　法

以 ANSI C 格式声明函数(由 C++ 维护)的一般形式是:

returnType functionName(typedParameterList)

示　例:

```
double sqr(double y)
{ return y * y; }

char prevChar(char c)
{ return c - 1; }
```

声明 C++ 函数时要牢记以下规定:

1. C++ 函数的返回类型应位于函数名的前面。
2. 如果参数表为空,可以用空括弧。在 C++ 中,也可以用 void 关键字显式地指明参数表为空。
3. 前面带类型的参数的列表组成了带类型的参数表,其一般格式如下:

   ```
   [const] type1 parameter1, [const] type2 parameter2, ...
   ```

 此格式表明每个参数就像变量一样被声明——先指出参数类型,然后是参数的标识符。在 C++ 中,参数表由逗号隔开。另外,不能将具有相同数据类型的一串参数组合在一起,必须显式地声明每个参数。如果参数中带有 const 项,那么编译器将确保函数不改变此参数的变元。
4. C++ 函数体括在花括号{}中。闭合的花括号后没有分号。
5. C++ 支持用值或引用传递变元。缺省情况下,参数用值传递变元。因而,函数工作时用的是数据的副本,原来的数据保持不变。若要声明一个引用参数,则将 & 字符插入到参数的数据类型的后面。引用参数成了其变元的别名,对此引用参数所作的任何改变都会影响此变元。引用参数的一般格式如下:

   ```
   [const] type1& parameter1, [const] type2& parameter2, ...
   ```

 如果参数中带有 const 项,那么编译器将确保函数不改变此参数的变元。
6. C++ 支持局部常量、数据类型和变量。虽然这些数据项可以出现在嵌套数据块语句中,但 C++ 不支持嵌套函数。
7. return 关键字返回函数值。
8. 如果函数的返回类型是 void,则无需使用 return 关键字,除非需要在函数中间提供出口。

新　词

定义原型(prototyping)

C++ 要求你在使用变量前或者声明或者定义一个函数。声明函数通常被称为定义原型,它列出函数名、返回值类型以及参数的个数及其类型。参数名这一项是可任选的。封闭的括弧后必须有一分号。如果在定义函数前调用函数,那么就需要声明函数。

以下是一个定义原型的简单例子:

```
// prototype the function square
double sqr(double);

main()
{
  cout << "5^ 2 = " << sqr(5) << "\n";
  return 0;
}

double sqr(double z)
{ return z * z; }
```

注意函数 sqr 的声明中仅包含其参数的类型。

一般说来,函数的声明是全局的。也可以在被调用的函数体内定义函数原型,这样就可以对其他函数隐藏其原型。

调用一个函数需要用变元提供其参数。这些变元按照参数声明的顺序分别与各参数对应。变元的数据类型必须与参数类型匹配或相容。例如,函数 volume 是这么定义的:

```
double volume(double length, double width, double height)
{
  return length * width * height;
}
```

要调用函数 volume 则需要 double 类型的或相容类型的变元(此处指所有数值数据类型)。以下是对函数 volume 的一些示范调用:

```
double len = 34, width = 55, ht = 100;
int i = 3;
long j = 44;
unsigned k = 33;

cout << volume(len, width, ht) << "\n";
cout << volume(1, 2, 3) << "\n";
cout << volume(i, j, k) << "\n";
cout << volume(len, j, 22.3) << "\n";
```

注　意　C++ 中允许放弃函数的返回结果。当我们关心的是函数做了什么而不是它返回什么值时就可用这种函数调用方式。

2.7　函数中的局部变量

良好的结构化程序设计技术强调函数应尽可能独立和可复用,因此函数可以有自己的数据类型、常量和变量以保持自己的独立性。

新　词

局部变量(local variable)

函数中的局部变量仅在宿主函数被调用时才存在。一旦函数终止,运行系统就将局部变量删掉。因而,局部变量的数据将在函数调用之间丢失。此外,运行系统在宿主函数每次被调用时都要对局部变量进行初始化。

要用局部变量存储并改变声明时带 const 项的参数的值。

不要声明与函数中要访问的全局变量名字相同的局部变量。

我们来看一个例子。清单 2.4 中显示了以下数学函数在自变量为 3.5 时的值和斜率:

$f(X) = X^2 - 5X + 10$

程序是用以下的近似公式计算斜率的:

$f'(X) = (f(X + h) - f(X - h)) / 2h$

这里, h 是一个小的增量。

键　入　　清单 2.4　程序 VAR2.CPP 的源代码

```
1: // C++ program that illustrates local variales in a function
2:
3: # include < iostream.h>
4:
5: double f(double x)
6: {
7: return x * x - 5 * x + 10;
8: }
9:
10: double slope(double x)
11: {
12: double f1, f2, incrim = 0.01 * x;
13: f1 = f(x + incrim);
14: f2 = f(x - incrim);
15: return (f1 - f2) / 2 / incrim;
16: }
17:
18: main()
19: {
20: double x = 3.5;
21:
22: cout < < "f(" < < x < < ") = " < < f(x) < < "\n"
23: < < "f'(" < < x < < ") = " < < slope(x) < < "\n";
24:
25: return 0;
26: }
```

以下是清单 2.4 中程序的一个运行范例:

输　出

```
f(3.5) = 4.75
f'(3.5) = 2
```

分　析

清单 2.4 中声明了三个函数,即 f(第 5 行),slope(第 10 行)和 main(第 18 行)。函数 f 很简单,它返回数学函数的值。函数 f 中未用到局部变量,而函数 slope 声明了局部变量 f1, f2 和 incrim。此函数也对后者进行初始化。第 13 行将 f(x+ incrim)的值赋给局部变量 f1。第 14 行将 f(x - incrim)的值赋给局部变量 f2。第 15 行使用局部变量 f1, f2 和 incrim 返回函数 slope 的值。main 函数只是显示一下当 x= 3.5 时数学函数及其斜率的值。

2.8　函数中的静态变量

清单 2.4 中,一旦函数 slope 终止,其中的局部变量也将丢失其值。C+ + 中可以将局部变量声明为静态的,这只需将关键字 static 放在其数据类型的前面即可。静态变量通常是要初始化的。初始化操作只进行一次,那是在宿主函数第一次被调用时进行的。

新　词

静态变量(static variable)

许多程序设计方法要求在函数调用之间保持变量的值。这些特殊的局部变量称为静态变量。

当宿主函数终止时,静态变量保持它们的值。编译器通过将静态变量存储在程序运行时所维护的一块独立的内存区来支持这一语言特性。可以在不同的函数中使用相同的静态变量名。这种雷同不会使编译器产生混淆,因为它会对哪个函数拥有哪些静态变量进行跟踪。

我们来看一个简单的程序。清单 2.5 中用一带静态变量的函数维护一个动态平均数。程序自己提供数据并数次调用此函数以获取并显示动态平均数的当前值。

键　入　　清单 2.5　程序 STATIC1.CPP 的源代码

```
1: // C+ +  program that illustrates static local variables
2:
3: # include < iostream.h>
4:
5: double mean(double x)
6: {
7: static double sum =  0;
8: static double sumx =  0;
9:
10: sum =  sum +  1;
11: sumx =  sumx +  x;
12: return sumx / sum
13: }
14:
15: main()
16: {
17: cout < <  "mean =  " < <  mean(1) < <  "\n";
```

```
18: cout << "mean = " << mean(2) << "\n";
19: cout << "mean = " << mean(4) << "\n";
20: cout << "mean = " << mean(10) << "\n";
21: cout << "mean = " << mean(11) << "\n";
22: return 0;
23: }
```

以下是清单 2.5 中程序的一个运行范例:

输　出

```
mean = 1
mean = 1.5
mean = 2.33333
mean = 4.25
mean = 5.6
```

分　析

清单 2.5 中的程序声明了包含静态变量的函数 mean。第 7 行和第 8 行分别声明了静态变量 sum 和 sumx。函数将这两个静态变量都初始化为 0。第 10 行的语句将变量 sum 值加 1。第 11 行的语句将变量 sumx 的值加 x。第 12 行返回用 sumx 除以 sum 所得到的动态平均数的值。

main 函数对函数 mean 进行了一连串的调用。第 17 行到第 21 行的流输出语句显示了更新的动态平均数。这些结果得来的可能值要归功于函数 mean 中的静态局部变量 sum 和 sumx。如果 C++ 中不支持静态局部变量,就得借助于使用全局变量——那便是一种很成问题的编程选择了!

2.9　嵌 入 函 数

使用函数时需要一些额外开销,即调用它们、传递其参数并返回其结果。在 C++ 中可以使用直接在其语句中展开的嵌入函数(inline function)。可见,嵌入函数可加快执行速度——尤其是当速度很关键的时候——其代价是展开了代码。

语　法

嵌入函数

嵌入函数的一般语法是:

```
inline returnType functionName(typedParameterList)
```

示　例:

```
inline double cube(double x)
{ return x * x * x; }

inline char nextChar(char c)
{ return c + 1; }
```

与使用嵌入函数作用相仿的另一种选择是使用# define 指令建立基于宏的伪函数。

许多 C+ + 程序员(包括笔者在内)竭力建议废弃这种方法而改用嵌入函数,产生这种倾向是因为嵌入函数提供了类型检查,而用# define 指令建立的宏则做不到这一点。

> 要在开发程序的开始阶段将嵌入函数声明为普通函数。因为非嵌入函数更容易调试。一旦程序调通了,再在需要的地方插入 inline 关键字。

> 不要声明包含过多语句的嵌入函数,增加. EXE 程序大小是不可取的。

以下是使用嵌入函数的一个简单示范程序。清单 2. 6 中含程序 INLINE1. CPP 的源代码。程序提示输入一个数,然后计算并显示输入数的平方和立方值。

键　入　　**清单 2. 6　程序 INLINE1. CPP 的源代码**

```
1: // C+ +  program that illustrates inline functions
2:
3: # include < iostream. h>
4:
5: inline double sqr(double x)
6: {
7:   return x *  x;
8: }
9:
10: inline double cube(double x)
11: {
12: return x *  x *  x;
13: }
14:
15: main( )
16: {
17:   double x;
18:
19:   cout < <  "Enter a number: ";
20:    cin > >  x;
21:
22:   cout < <  "Square of " < <  x < <  " = " < <  sqr(x) < <  "\n"
23:         < <  "cube of " < <  x < <  " = " < <  cube(x) < <  "\n"
24:
25:   return 0;
26: }
```

以下是清单 2. 6 中程序的一个运行范例:

输　出

```
Enter a number: 2. 5
square of 2. 5 =  6. 25
cube of 2. 5 =  15. 625
```

分　析

清单 2.6 中的程序声明了嵌入函数 sqr 和 cube。每一函数标题都以关键字 inline 开始。嵌入函数的其他方面与一般的短函数相似。main 函数调用函数 sqr 和 cube, 分别显示平方值和立方值。

2.10　从函数中退出

由于某些特定条件不允许继续执行函数中的语句时通常要从函数中提前退出。C++ 中提供的 return 语句可实现从函数中退出。如果函数是 void 类型的, 那么可以用后面不带任何表达式的 return 语句。相反, 如果要从非 void 类型的函数中退出, 那么返回语句应产生一个值表示退出函数的目的。

2.11　缺 省 变 元

缺省变元是一种非常简单但非常有用的语言特性。如果不指定参数的变元但该参数具有缺省变元时, 变元会被自动使用。

新　词

缺省变元(default argument)
C++ 中可以用缺省变元作为函数的参数。

用缺省变元时要遵循以下规定:

1. 一旦用缺省变元作参数, 则同一参数表中后面的所有参数也必须这么做。不能随机地用缺省变元作参数。这条规定意味着参数表可被分成两个子表: 头上的参数不带缺省变元, 而尾部的参数带缺省变元。
2. 必须为每一个不带缺省变元的参数提供变元。
3. 对于具有缺省变元的参数, 可以省略其变元。
4. 一旦省略了带有缺省变元的参数的变元, 则所有后继参数的变元也必须省略。

注　意　排列带缺省变元的参数的最好的办法是按照使用其缺省变元的可能性来排列它们。将最不可能使用的变元放在第 1 个, 而最有可能使用的变元放在最后。

让我们来看一个使用带缺省变元的函数的简单例子。清单 2.7 中列出了程序 DEFARGS1.CPP 的源代码。程序提示输入两点的 X 和 Y 坐标, 然后计算并显示两点之间以及每个点到原点的距离。

键　入　清单 2.7　程序 DEFARGS1.CPP 的源代码

```
1: // C++ program that illustrates default arguments
```

```
2:
3: # include < iostream.h>
4: # include < math.h>
5:
6: inline double sqr(double x)
7: { return x * x; }
8:
9: double distance (double x2, double y2,
10:                 double x1 = 0, double y1 = 0)
11: {
12: return sqrt(sqr(x2 - x1) + sqr(y2 - y1));
13: }
14:
15: main()
16: {
17:   double x1, y1, x2, y2;
18:
19:   cout < < "Enter x coordinate for point 1: ";
20:   cin > > x1;
21:   cout < < "Enter y coordinate for point 1: ";
22:   cin > > y1;
23:   cout < < "Enter x coordinate for point 2: ";
24:   cin > > x2;
25:   cout < < "Enter y coordinate for point 2: ";
26:   cin > > y2;
27:
28:   cout < < "distance between points = "
29:        < < distance(x1, y1, x2, y2) < < "\n";
30:   cout < < "distance between point1 and (0, 0) = "
31:        < < distance(x1, y1, 0) < < "\n";
32:   cout < < "distance between point2 and (0, 0) = "
33:        < < distance(x2, y2) < < "\n";
34:
35:   return 0;
36: }
```

以下是清单 2.7 中程序的一个运行范例:

输　出

```
Enter x coordinate for point 1: 1
Enter y coordinate for point 1: 1
Enter x coordinate for point 2: - 1
Enter y coordinate for point 2: 1
distance between points = 2
distance between point 1 and (0, 0) = 1.41421
distance between point 2 and (0, 0) = 1.41421
```

分　析

清单 2.7 中的程序所包含的是 2 个而不是 1 个头文件。第 4 行用# include **指令包含了声明平方根数学函数** sqrt **的头文件** MATH.H**。程序在第 6 行声明了嵌入函数** sqr**, 此**

函数返回参数 x 所对应变元的平方值。参数 x2 和 y2 分别代表第 2 个点的 X 和 Y 坐标，而参数 x1 和 y1 分别代表第 1 个点的 X 和 Y 坐标。x1 和 y1 这两个参数都带有缺省的变元 0。函数返回这两个点之间的距离。如果省略 x1 和 y1 的变元，那么函数返回(x2，y2)和原点(0，0)之间的距离。如果仅仅省略最后一个参数的变元，则函数计算(x2，y2)和(x1，0)这两点之间的距离。

main 函数中，第 19 到 26 行的语句提示输入两个点的 X 和 Y 坐标；第 28 到 29 行以 4 个变元即 x1，y1，x2 和 y2 调用函数 distance，因此，这次调用函数 distance 未用缺省变元；相反地，第 30 行到 31 行的语句只用三个变元调用函数 distance，这次调用函数使用了最后一个参数的缺省变元；第 32 到 33 行的语句仅用了两个变元调用函数 distance，这次调用用到了第 3 和第 4 个参数的缺省变元。第 2 次调用 distance 时也可以省略第 3 个变元，程序仍然能通过编译，正常运行。

2.12　函 数 重 载

函数重载是 C+ + 的语言特性，这是 C，Pascal 或 BASIC 所不具有的。用这一新特性可以声明具有相同名字但不同参数表的多个函数。函数的返回类型并不是函数特征的一部分，因为 C+ + 允许你放弃返回类型。

新　词

函数特征(function signature)
参数表也被称为函数特征。

警　告　在重载函数中使用缺省变元可能会使一些函数的特征(当用到缺省变元时)产生重复。C+ + 编译器能够检测到此二义性并产生编译错误。

要用缺省变元减少重载函数的数量。

不要用重载函数实施不同的操作。

我们来看一个使用重载函数的简单例子。清单 2.8 中列出了程序 OVERLOAD.CPP 的源代码。程序完成以下操作：

- 声明了具有 char，int 和 double 类型的变量并初始化其值
- 显示初始值
- 调用重载函数并为变量增值
- 显示更新后的变量值

键　入　清单 2.8　程序 OVERLOAD.CPP 的源代码

```
1: // C+ + program that illustrates function overloading
2:
```

```
3: # include < iostream.h>
4:
5: // inc version for int types
6: void inc(int& i)
7: {
8:   i = i + 1;
9: }
10:
11: // inc version for double types
12: void inc(double& x)
13: {
14:   x = x + 1;
15: }
16:
17: // inc version for char types
18: void inc(char& c)
19: {
20: c = c + 1;
21: }
22:
23: main()
24: {
25:   char c = 'A';
26:   int i = 10;
27:   double x = 10.2;
28:
29:   // display initial values
30:   cout < < "c = " < < c < < "\n"
31:        < < "i = " < < i < < "\n"
32:        < < "x = " < < x < < "\n"
33:   // invoke the inc functions
34:   inc(c);
35:   inc(c);
36:   inc(c);
37:   // display updated values
38:   cout < < "After using the overloaded inc function\n";
39:   cout < < "c = " < < c < < "\n"
40:   cout < < "i = " < < i < < "\n"
41:   cout < < "x = " < < x < < "\n"
42:
43:   return 0;
44: }
```

以下是清单 2.8 中程序的一个运行范例:

输　出

```
c = A
i = 10
x = 10.2
After using the overloaded inc function
c = B
i = 11
```

```
x = 11.2
```

清单 2.8 中的程序声明了重载 void 函数 inc 的三个版本。函数 inc 的第 1 个版本带有引用型参数 i, 该函数将参数 i 的值加 1。因为 i 是对其变元的引用, 函数 inc(int&) 的动作会影响到在函数体外的变元。函数 inc 的第 2 个版本带有 double 型的参数 x, 该函数也将参数值增 1。因为 x 是对其变元的引用, 函数 inc(double&) 的动作会影响到在函数体外的变元。函数 inc 的第 3 个版本带有 char 型的引用参数 c, 该函数也将参数值增 1。因为 c 是对其变元的引用, 引用参数会影响到在函数体外的变元。

main 函数声明的变量 c, i 和 x 分别具有 char, int 和 double 类型, 该函数还分别用值 'A', 10 和 10.2 初始化变量 c, i 和 x。第 30 到 32 行的语句显示了变量 c, i 和 x 的初始值。main 函数在第 34 到 36 行调用重载函数。第 34 行中对 inc 的调用产生对函数 inc (char&)的调用, 因为所用的变元是 char 型的变量。第 35 行中对函数 inc 的调用导致对函数 inc(int&)的调用, 因为所用变元类型是 int 型的。第 36 行中对函数 inc 的调用将产生对函数 inc(double&)的调用, 因为所用变元是 double 型变量。第 39 行到第 41 行的输出语句显示了变量 c、i 和 x 的值。

小　　结

今天这一课讲述了 C+ + 程序的基本组成部分。这些部分包括数据类型、变量、常量和函数。基本内容可概括如下:

- Visual C+ + 中的预定义数据类型包括 int, char, float, double 和 void 数据类型。C+ + 通过支持数据类型修饰符增加了数据类型的灵活性。这些修饰词改变数值的精度和范围。类型修饰符是 signed, unsigned short 和 long。
- Visual C+ + 标识符最长可达 247 个字符并且必须以字母或下划线开始, 后面跟以字母、数字或下划线。C+ + 标识符是大小写相关的。
- # include 指令是针对编译器的特殊命令。该指令让编译器包含指定文件的内容, 就好像在当前扫描的文件中键入它们一样。
- 声明变量时需要指明变量的数据类型和变量名。C+ + 允许在声明变量时进行初始化。可以在一个声明语句中声明多个变量。
- 声明变量包括用# define 指令声明基于宏的常量或使用 const 关键字声明正规常量。正规常量要求指定常量的类型(缺省时为 int)、常量名及相关值。
- 定义函数的一般格式是:

```
returnType functionName(parameterList)
{
< declaration of data items>

< function body>
return returnValue;
}
```

如果函数在定义之前要被客户函数调用, 那么必须先指定其原型。指定函数

原型的一般格式为:

```
returnType functionName(parameterList);
```

参数表中可以省略参数的名字。

- 函数中的局部变量支持实现函数的高度独立性。声明局部变量的方法与声明全局变量相似。
- 静态变量的声明方法是将关键字 static 置于变量的数据类型的前面。静态变量在函数调用之间保持其值。在多数情况下,需要初始化静态变量。这些初始值在程序首次调用宿主函数时被赋给静态变量。
- 嵌入函数可就地展开语句,就像基于宏的伪函数一样。但是,与伪函数不一样的是,嵌入函数要进行类型检查。
- 用 return 语句可退出函数。void 函数中不需要在 return 语句后带表达式。
- 可以用缺省变元为函数的参数指定缺省值。如果省略参数的变元,而该参数具有缺省变元时,该变元被自动使用。
- 函数重载使你可以定义多个具有相同名字但参数表不同(也称为函数特征)的函数。函数的返回类型不是函数特征的一部分,因为 C++ 中允许放弃返回类型。

问 与 答

问: 命名标识符是否有一些特定风格?

答: 近几年确有一些流行的风格。笔者所用的一种是标识符以小写字符开始,如果其中有多个单词,例如 numberOfElements,那么后面所跟的每一个单词的第一个字符都用大写。

问: C++ 函数中可以声明嵌套函数吗?

答: 不能。嵌套函数在运行时会增加不少额外开销。

问: 什么时候可以使用静态全局变量?

答: 任何时候都不可以。全局变量不需要声明为静态的,因为它们存在于整个程序的生命期。

作 业

作业中所提供的测验题帮助你巩固所学内容,而练习则就使用所学内容提供一些经验。要弄懂测验和练习的解答后再继续下一天的课程。附录 B 的“答案”中提供了解答。

测 验

1. 以下变量中哪些是合法的,哪些是非法的(为什么)?

```
numFiles
```

```
n0Distance- 02- Line
0Weight
Bin Number
static
Static
```

2. **以下程序的输出是什么？对于函数 swap 你有什么看法？**

```
# include < iostream.h>

void swap(int i, int j)
{
  int temp = 1;
  i = j;
  j = temp;
}

main()
{
int a = 10, b = 3;
swap(a, b);
cout < < "a = " < < a < < " and b = " < < b;
return 0;
}
```

3. **以下程序的输出是什么？对于函数 swap 你有什么看法？**

```
# include < iostream.h>

void swap(int& i, int& j)
{
int temp = 1;
i = j;
j = temp;
}

main()
{
  int a = 10, b = 3;
  swap(a, b);
  cout < < "a = " < < a < < " and b = " < < b;
  return 0;
}
```

4. **下面的重载函数有什么问题？**

```
void inc(int& i)
{
  i = i + 1;
}

void inc(int& i, int diff = 1}
{
  i = i + diff;
}
```

5. **下面的函数中有什么错误?**

```
double volume(double length, double width = 1, double height)
{
return length * width * height
}
```

6. **下面的函数中有什么错误?**

```
void inc(int& i, int diff = 1)
{
  i = I + diff;
}
```

7. **下面的程序中有什么错误,如何改正?**

```
# include < iostream.h>
main()
{
  double x = 5.2;

  cout < < x < < "^ 2 = " < < sqr(x);
  return 0;
}

double sqr(double x)
{ return x * x; }
```

练　习

在 OVERLOAD.CPP **基础上建立新程序** OVERLOAD2.CPP,**给重载函数** inc **加上第** 2 **个参数,使其具有缺省变元,新参数应代表增量值,缺省变元为** 1。

第3天

运算符和表达式

对数据的运算要用到表达式,它是由操作数和运算符组成的。C++ 支持多种运算符和表达式。

新 词

运算符(operator)　　操作数(operand)
运算符是用操作数的值产生出新的值的特殊符号。

每种运算符都以特定方法进行数据运算。今天要学习以下专题:

- 算术运算符和表达式
- 增量运算符
- 算术赋值运算符
- 类型强制和数据转换
- 关系运算符和条件表达式
- 位操作运算符
- 逗号运算符

3.1 算术运算符

表 3.1 中列出了 C++ 的运算符。编译器根据操作数决定是进行浮点除法还是整数除法。如果两个操作数都是整数表达式,则编译器产生整数除法代码;如果有一个或两个操作数是浮点表达式,则编译器生成浮点除法代码。

表 3.1 C++ 算术运算符

C++ 运算符	用 途	数据类型	举 例
+	单目加	数值	x = + y + 3;
-	单目减	数值	x = - y;
+	加法	数值	z = y + x;
-	减法	数值	z = y - x;
*	乘法	数值	z = y * x;
/	除法	整数	z = y / x;
%	取模	整数	z = y % x;

我们来看一个对整数和浮点数使用数学运算符的例子。清单 3.1 中列出了程序 OPER1.CPP 的源代码。该程序完成以下操作:

- 提示输入两个整数(每次提示输入一个);
- 对这两个整数使用+，-，*，/和%运算符,将结果存在单独的变量中;
- 显示整数运算的结果;
- 提示输入两个浮点数(每次提示输入一个);
- 对这两个整数使用+，-，* 和/运算符,将结果存在单独的变量中;
- 显示浮点运算的结果;

键　入　　清单 3.1　程序 OPER1.CPP 的源代码

```
1:  // simple C++ program to illustrate simple math operations
2:
3:  #include <iostream.h>
4:
5:  main()
6:  {
7:
8:     int int1, int2;
9:     long long1, long2, long3, long4, long5;
10:    float x, y, real1, real2, real3, real4;
11:
12:    cout << "\nType first integer : ";
13:    cin >> int1;
14:    cout << "Type second integer : ";
15:    cin >> int2;
16:    cout << "\n";
17:    long1 = int1 + int2;
18:    long2 = int1 - int2;
19:    long3 = int1 * int2;
20:    long4 = int1 / int2;
21:    long5 = int1 % int2;
22:    cout << int1 << "+ " << int2 << "= " << long1 << "\n";
23:    cout << int1 << "- " << int2 << "= " << long2 << "\n";
24:    cout << int1 << "* " << int2 << "= " << long3 << "\n";
25:    cout << int1 << "/ " << int2 << "= " << long4 << "\n";
26:    cout << int1 << "mod " << int2 << "= " << long5 << "\n";
27:    cout << "\n\n";
28:    cout << "Type first real number : ";
29:    cin >> x;
30:    cout << "Type second real number : ";
31:    cin >> y;
32:    cout << "\n";
33:    real1 = x + y;
34:    real2 = x - y;
35:    real3 = x * y;
36:    real4 = x / y;
37:    cout << x << "+ " << y << "= " << real1 << "\n";
38:    cout << x << "- " << y << "= " << real2 << "\n";
39:    cout << x << "* " << y << "= " << real3 << "\n";
40:    cout << x << "/ " << y << "= " << real4 << "\n";
41:    cout << "\n\n";
```

```
42:       return 0;
43:    }
```

以下是清单 3.1 中程序的一个运行范例。

输　出

```
Type first integer : 10
Type second integer : 5

10 + 5 = 15
10 - 5 = 5
10 * 5 = 50
10 / 5 = 2
10 mod 5 = 0

Type first real number : 1.25
Type second real numbe : 2.58

1.25 + 2.58 = 3.83
1.25 - 2.58 = - 1.33
1.25 * 2.58 = 3.225
1.25 / 2.58 = 0.484496
```

分　析

清单 3.1 中的程序在函数 main 中声明了一组 int 型, long 型和 float 型的变量。其中一些存放输入内容而另一些存放数学运算的结果。第 12 行的输出语句提示你输入第 1 个整数。第 13 行的输入语句获取输入并将其存放在变量 int1 中。第 14 和 15 行进行类似的操作: 提示输入第 2 个整数并将其存于变量 int2 中。

程序在第 17 到第 21 行进行整数数学运算并将这些运算的结果存在变量 long1 到 long5 中。将这些变量声明为 long 型是为了防止数值可能溢出。第 22 到 26 行的输出语句显示整数操作数、所用的运算符以及结果。

第 28 行的输出语句提示你输入第 1 个浮点数。第 29 行的输入语句获取输入并将其存于变量 x 中。第 30 和 31 行进行类似的操作: 提示输入第 2 个浮点数并将其存于变量 y 中。

新　词

实数(real number)
浮点数也叫实数。

程序在第 33 到 36 行进行浮点数学运算并将这些运算的结果存于变量 real1 到 real4 中。第 37 到 40 行的输出语句显示操作数、所用运算符以及结果。

3.2　算术表达式

最简单的一类表达式仅仅包含文字,例如:

```
-12
34.45
'A'
"Hello"
```

新　词

算术表达式(arithmetic expression)
一般来说,算术表达式是程序语句中包含值的那一部分。

文字常量-12 和 35 是最简单的算术表达式,再下一级的算术表达式是单个的变量或常数,例如:

```
DAYS_IN_WEEK    // a constant
i
x
```

更下一级的算术表达式含一个运算符,操作数为数值、常数或变量。这里举几个例子:

```
355 / 113
4 * i
45.67 + x
```

更高级的算术表达式中含多个运算符、括弧甚至函数,如:

```
(355/113) * square(radius)
PIE * square(radius)
((2 * x - 3) * x + 2) * x - 5
(1 + x) / (3 - x)
```

介绍完其他类型的运算符后,在这一课结束时将讨论执行运算符的优先级。

3.3　增量运算符

C++ 支持专门的增量和减量运算符。

新　词

增量和减量运算符(increment operators, decrement operators)
用增量(++)和减量(--)运算符可以分别将变量的值增 1 或减 1。

语　法

增量运算符
增量运算符的一般语法是:

```
variable++    // post-increment
++variable    // pre-increment
```

示　例:

```
lineNumeber++;
++index;
```

语　法

减量运算符

减量运算符的一般语法是:

```
variable - -    // post-decrement
- - variable    // pre-decrement
```

示　例:

```
lineNumber - - ;
- - index;
```

这些一般语法格式表明有两种使用+ + 和- - 运算符的方法。如果将运算符放在其操作数的左边,则在操作数将其值交给表达式之前改变其值;而如果将运算符放在其操作数的右边,则在运算符将其值交给表达式之后才改变其值。如果语句中仅含+ + 和- - 运算符,那么用前、后格式实际上没有区别。

以下是一些简单的例子:

```
int n, m, t = 5;
t+ + ; // t is now 6, same effect as + + t
- - t; // t is now 5, same effect as t- -
t = 5;
n = 4 * t+ + ; // t is now 6 and n is 20
m = 4 * + + t; // t is new 6 and m is 24
```

第 1 条语句使用后增量+ + 运算符增加变量 t 的值。如果改用+ + t,则语句执行结束后会得到同样的结果。第 2 条语句使用了前减量- - 运算符。如果改用 t- - 那么也会得到相同的结果。下一条语句将 5 赋给变量 t 然后在一简单数学表达式中使用后增量+ + 运算符。该语句用 4 乘以 t 的当前值(即 5),将结果 20 赋给变量 n,然后将变量 t 的值增加到 6。最后两条语句显示了不同的结果。最后一条语句先将变量 t 增值(t 的值变为 6),然后进行乘法,最后将结果 24 赋给变量 m。

我们来看一个简单的程序,它示意了增量运算符的特性。清单 3.2 中列出了程序 OPER2.CPP 的源代码。该程序无需任何输入,它只是显示由增量运算符求出的两个整数值。

键　入　　**清单 3.2　程序 OPER2.CPP 的源代码**

```
1:  /*
2:  C+ + program to illustrate the feature of the increment operator.
3:  The + + or - - may be included in an expression. The value
4:  of the associated variable is altered after the expression
5:  is evaluated if the var+ + (or var- - ) is used, or before
6:  when + + var (or - - var) is used.
7:  * /
8:
9:  # include < iostream.h>
10:
11: main()
12: {
```

```
13:   int i, k = 5;
14:
15:   // use post- incrementing
16:   i = 10 * (k++); // k contributes 5 to the expression
17:   cout << "i = " << i << "\n\n"; // displays 50 (= 10 * 5)
18:
19:   k--; // restores the value of k to 5
20:
21:   // use pre- incrementing
22:   i = 10 * (++k); // k contributes 6 to the expression
23:   cout << "i = " << i << "\n\n"; // displays 60 ( = 10 * 6)
24:   return 0;
25: }
```

以下是清单 3.2 中程序的一个运行范例:

输　出

```
i = 50
i = 60
```

分　析

清单 3.2 中的程序在函数 main 中声明了两个 int 型的变量,即 i 和 k。函数初始化变量 k,为其赋值- 5。第 16 行的语句对变量 k 使用后增量运算符。接着,语句用 10 乘以 k 的初始值 5 并将结果 50 赋给变量 i,此后,程序将变量 k 增值。第 17 行的输出语句显示变量 i 的值。第 19 行的语句将变量 k 的值减回到 5。第 22 行的语句对变量 k 运用前增量运算符。因此,程序首先将变量 k 增值(从 5 到 6),然后用 10 去乘以更新后的 k 值。程序将乘法的结果 60 赋给变量 i。第 23 行的输出语句显示变量 i 的值。

3.4　赋值运算符

程序员经常会遇到这样的语句:

```
IndexOfFirstElement = IndexOfFirstElement + 4;
GraphicsScaleRatio = GraphicsScaleRatio * 3;
CurrentRateOfReturn = CurrentRateOfReturn / 4;
DOSfileListSize = DOSfileListSize - 10;
```

接收表达式结果的变量同时也是第 1 操作数。(当然,加法和乘法是可交换运算符。因而对这些操作数来说,被赋值的变量可以是两个操作数中的任一个。)请注意笔者选用了相对较长的名字以提醒你要缩短表达式的长度,但不要缩短变量名的长度。

新　词

赋值运算符(assignment operator)

C++ 中提供由简单数学运算符合并而成的赋值运算符。

可以写出以下语句:

```
IndexOfFirstElement + = 4;
GraphicsScaleRatio * = 3;
CurrentRateOfReturn / = 4;
DOSfileListSize - = 10;
```

请注意变量名仅出现一次,另外语句中使用了+ = , * = , / = 和- = 运算符。表 3.2 中列出了算术赋值运算符。C+ + 中还支持另外一些类型的赋值运算符。

表 3.2　算术赋值运算符

赋值运算符	长表达式	举　例
x + = y	x = x + y	x + = 12;
x - = y	x = x - y	x - = 34 + y;
x * = y	x = x * y	scale * = 10;
x / = y	x = x / y	z / = 34 * y;
x % = y	x = x % y	z % = 2;

我们来看一个程序,其中对整数和浮点数运用了赋值运算符。清单 3.3 中列出了程序 OPER3.CPP 的源代码。该程序完成以下操作:

- 提示输入两个整数(每次提示输入一个);
- 对这两个整数运用一组赋值和增量运算符;
- 显示整数的新值;
- 提示输入两个浮点数(每次提示输入一个);
- 对这两个数运用一组赋值和增量运算符;
- 显示浮点数的新值。

键　入　　清单 3.3　程序 OPER3.CPP 的源代码

```
1:  // C++ program to illustrate math assignment operators
2:
3:  # include < iostream.h>
4:
5:  main()
6:  {
7:     int i, j;
8:     double x, y;
9:
10:    cout < < "Type first integer : ";
11:    cin > > i;
12:    cout < < "Type second integer : ";
13:    cin > > j;
14:    i + = j;
15:    j - = 6;
16:    i * = 4;
```

```
17:     j /= 3;
18:     i++;
19:     j--;
20:     cout << "i = " << i << "\n";
21:     cout << "j = " << j << "\n";
22:
23:     cout << "Type first real number: ";
24:     cin >> x;
25:     cout << "Type second real number: ";
26:     cin >> y;
27:     // abbreviated assignments also work with doubles in C++
28:     x += y;
29:     y -= 4.0;
30:     x *= 4.0;
31:     y /= 3.0;
32:     x++;
33:     y--;
34:     cout << "x = " << x << "\n";
35:     cout << "y = " << y << "\n";
36:     return 0;
37: }
```

以下是清单 3.3 中程序的一个运行范例:

输　出

```
Type first integer: 55
Type second integer: 66
i = 485
j = 19
Type first real number: 2.5
Type second real number: 4.58
x = 29.32
y = -0.806667
```

分　析

清单 3.3 中的程序在 main 函数内的第 7 行和第 8 行分别声明了两个 int 型的变量(i 和 j)以及两个 double 型变量(x 和 y)。第 10 行的输出语句提示你输入第 1 个整数。第 11 行的输入语句接收输入并将其存于变量 i 中。第 12 和 13 行与第 10 和 11 行类似: 提示输入第 2 个整数并将其存于变量 j 中。

程序用第 14 到 19 行的语句操作变量 i 和 j 的值。在第 14 行中, 程序用+= 运算符将变量 i 的值加上变量 j 的值。第 15 行用-= 运算符将变量 j 的值减去 6。第 16 行用*= 运算符将变量 i 的值乘以 4 并将结果回赋给 i。第 17 行利用/= 运算符将变量 j 的值除以 3 并将结果存到 j 中。第 18 和 19 行分别对变量 i 和 j 使用增量和减量运算符。第 20 行和 21 行的输出语句分别显示变量 i 和 j 的内容。

第 23 行的输出语句提示你输入第 1 个浮点数。第 24 行的输入语句接收输入并将其

存于变量 x 中。第 25 和 26 行类似于第 23 和 24 行——提示你输入第 2 个浮点数并将其存于变量 y 中。

程序用第 28 到 33 行的语句操作变量 x 和 y 的值。第 28 行中，程序用+ = 运算符将变量 x 的值加上变量 y 的值。第 29 行用- = 运算符将变量 y 的值减 4。第 30 行用* = 运算符将变量 x 的值乘以 4 并将结果送回到 x 中。第 31 行利用/ = 运算符将 y 的值除以 3 并将结果存到 y 中。第 32 和 33 行分别对变量 x 和变量 y 运用增量和减量运算符。第 34 和 35 行中的输出语句分别显示变量 x 和 y 的值。

3.5　sizeof 运算符

程序设计时经常需要知道数据类型或变量的字节数。C+ + 中提供了 sizeof 运算符，它以数据类型或变量作为变元。

语　法

sizeof 运算符

sizeof 运算符的一般语法是：

```
sizeof ( {variable- name ©¦data- type})
sizeof {variable- name ©¦data- type}
```

示　例：

```
int sizeDifference =  sizeof(int) -  sizeof(float);
int intSize =  sizeof int;
```

> 要对变量名而不是其数据类型用 sizeof。这种方法是最安全的。因为如果你改变了变量的数据类型，sizeof 运算符仍将返回正确的答案；相反，如果对变量的数据类型用了 sizeof 运算符且后来又改变了变量的数据类型，那么若不更新 sizeof 运算符的变元则会产生错误。
>
> ---
>
> 不要用数值表示变量的存储大小，这种方法经常会导致错误发生。

我们来看一个对变量和数据类型使用了 sizeof 运算符的例子。清单 3.4 中列出了程序 SIZEOF1.CPP 的源代码。程序显示了两个相同的表，它们都示意了 short int，int，long int，char 和 float 数据类型的大小。程序通过对上述类型变量运用 sizeof 运算符来显示第 1 个表；通过直接对上述类型运用 sizeof 运算符显示第 2 个表。

键　入　　清单 3.4　程序 SIZEOF1.CPP 的源代码

```
1:  /*
2:  simple program that returns the data sizes using the sizeof( )
3:  operator with variables and data types.
4:  * /
5:
```

```
#include <iostream.h>

main()

{
  short int aShort;
  int anInt;
  long aLong;
  char aChar;
  float aReal;

  cout << "Table 1. Data sizes using sizeof(variable)\n\n";
  cout << "    Data type      Memory used\n";
  cout << "                   (bytes)\n";
  cout << "- - - - - - - - - - - - - - - - -   - - - - - - - - - - - - - - -";
  cout << "\n      short int             " << sizeof(aShort);
  cout << "\n      integer               " << sizeof(anInt);
  cout << "\n      long integer          " << sizeof(aLong);
  cout << "\n      character             " << sizeof(aChar);
  cout << "\n      float                 " << sizeof(aReal);
  cout << "\n\n\n\n";

  cout << "Table 2. Data sizes using sizeof(dataType)\n\n";
  cout << "    Data type            Memory used\n";
  cout << "                       (bytes)\n";
  cout << "- - - - - - - - - - - - - - - - -   - - - - - - - - - - - - - - -";
  cout << "\n      short int         " << sizeof(short int);
  cout << "\n      integer           " << sizeof(int);
  cout << "\n      long integer      " << sizeof(long);
  cout << "\n      character         " << sizeof(char);
  cout << "\n      float             " << sizeof(float);
  cout << "\n\n\n\n";

  return 0;
}
```

以下是清单 3.4 中程序的一个运行范例:

输　出

```
Table 1. Data sizes using sizeof(variable)

    Data type                    Memory used
                                   (bytes)
- - - - - - - - - - - - - - - -   - - - - - - - - - - - -
      short int                        2
      integer                          2
    long integer                       4
      character                        1
      float                            4

Table 2. Data sizes using sizeof(dataType)
```

Data type	Memory used (bytes)
short int	2
integer	2
long integer	4
character	1
float	4

分　析

清单 3.4 中的程序在函数 main 中声明了 5 个变量，每个变量的数据类型都不相同且变量名是从其数据类型派生得来的。例如，变量 anInt 是一个 int 型变量，而 aLong 是一个 long 型变量，等等。

第 17 到 25 行的语句显示了数据大小表。第 21 到 25 行的语句对变量使用了 sizeof 运算符。

第 28 到 36 行的语句显示了数据大小表。第 32 到 36 行的语句对数据类型标识符使用 sizeof 运算符。

3.6　类 型 强 制

将数值从一种数据类型转换到另一种数据类型的自动数据转换是编译器的职责之一。这种数据转换可以简化表达式，这样无论是初学者还是熟练程序员都可减轻负担。有了幕后的数据转换工作，就不需要仔细检查程序中混合了相容数据类型的表达式。例如：编译器会处理大多数混合了不同类型的整数或混合了整数和浮点数的表达式。但是如果你企图干一些非法的事情则会出编译错误！

新　词

类型强制(typecasting)

类型强制是一种语言特性，用它可以显式地指定如何将一个值从其原始数据类型转换成另一种相容的数据类型。因而，类型强制指示编译器进行你想要的转换而不是编译器认为需要转换！

语　法

类型强制

C++ 支持以下格式的类型强制：

```
type_cast(expression)
```

和

```
(type_cast) expression
```

我们来看一个示意隐式数据转换和类型强制的例子。清单 3.5 中列出了程序 TYPECAST1.CPP 的源代码。程序中声明了字符型、整型和浮点型的变量，然后进行两组相似的数学运算。第 1 组依靠由编译器进行的数据类型的自动转换；第 2 组使用了类型强制显式指示编译器如何转换数据类型。

该程序无需任何输入——它自己提供数据——并且显示两组运算的输出值。程序表明编译器对这两组运算正确地产生了同一输出结果。

键　入　　**清单 3.5　程序 TYPCAST1.CPP 的源代码**

```
1:  // simple C++ program that demonstrates typecasting
2:
3:  #include <iostream.h>
4:
5:  main()
6:  {
7:  short shortInt1, shortInt2;
8:  unsigned short aByte;
9:  int anInt;
10: long aLong;
11: char aChar;
12: float aReal;
13:
14: // assign values
15: shortInt1 = 10;
16: shortInt2 = 6;
17: // perform operations without typecasting
18: aByte = shortInt1 + shortInt2;
19: anInt = shortInt1 - shortInt2;
20: aLong = shortInt1 * shortInt2;
21: aChar = aLong + 5; // conversion is automatic to character
22: aReal = shortInt1 * shortInt2 + 0.5;
23:
24: cout << "shortInt1 = " << shortInt1 << '\n'
25:      << "shortInt2 = " << shortInt2 << '\n'
26:      << "aByte = " << aByte << '\n'
27:      << "anInt = " << anInt << '\n'
28:      << "aLong = " << aLong << '\n'
29:      << "aChar is " << aChar << '\n'
30:      << "aReal = " << aReal << "\n\n\n";
31:
32: // perform operations with typecasting
33: aByte = (unsigned short) (shortInt1 + shortInt2);
34: anInt = (int) (shortInt1 - shortInt2);
35: aLong = (long) (shortInt1 * shortInt2);
36: aChar = (unsigned char) (aLong + 5);
37: aReal = (float) (shortInt1 * shortInt2 + 0.5);
38:
39: cout << "shortInt1 = " << shortInt1 << '\n'
40:      << "shortInt2 = " << shortInt2 << '\n'
41:      << "aByte = " << aByte << '\n'
42:      << "anInt = " << anInt << '\n'
43:      << "aLong = " << aLong << '\n'
44:      << "achar is " << aChar << '\n'
45:      << "aReal = " << aReal << "\n\n\n";
46: return 0;
47: }
```

以下是清单 3.5 中程序的一个运行范例:

输　出

```
shortInt1 =  10
shortInt2 =  6
aByte =  16
anInt =  4
aLong =  60
aChar is A
aReal =  60.5

shortInt1 =  10
shortInt2 =  6
aByte =  16
anInt =  4
aLong =  60
aChar is A
aReal =  60.5
```

分　析

清单 3.5 中的程序在函数 main 中声明了以下变量:

- short 型变量 shortInt1 和 shortInt2
- unsigned short 型变量 aByte
- int 型变量 anInt
- long 型变量 aLong
- char 型变量 aChar
- float 型变量 aReal

第 15 行和第 16 行分别将整数 10 和 16 赋给变量 shortInt1 和 shortInt2。第 18 到 22 行进行不同的数学运算并将结果赋给 aByte，anInt，aLont，aChar 和 aReal。

注　意　C 和 C++ 中将 char 型作为特殊的整数对待。每一 char 型文字(如 'A')、常量或变量都有一与其 ASCII 码相等的整数值。这一语法特性使得你可以将整数存于一个 char 型的变量中或将 char 型数据项作为整数对待。第 21 行的语句将变量 aLong 的值加 5 并将其结果——整数——赋给变量 aChar。被赋值的整数值 65 代表字母 A 的 ASCII 码。

第 24 到 30 行的输出语句显示了变量中存放的值。注意变量 aChar 的输出是字母 A。如果将变量 aChar 的输出语句格式写成< < (int)aChar,则得到的是 65,这是 aChar 中存放的字符 ASCII 码。

第 32 到 37 行的语句所进行的运算与第 18 到 22 行的类似。主要区别在于第 32 到 37 行的语句使用了类型强制显式指示编译器如何转换结果。第 39 到 45 行的输出语句显示

变量的内容。

3.7　关系和逻辑运算

表 3.3 中列出了关系和逻辑运算符。注意 C+ + 并不拼写运算符 AND, OR 和 NOT, 而是利用单字符或双字符符号。还要注意 C+ + 中不支持异或关系运算。可以用下面的# define 宏指令将 AND, OR 或 NOT 标识符定义为宏。

新　词

关系运算符(relational operators)　　**逻辑运算符**(logical operators)

关系运算符(大于、小于或等于)以及逻辑运算符(与、或、非)在任何程序设计语言中都是决策机构的基本要素。

```
# define AND &&
# define OR ©©¦¦
# define NOT ©¦¦
```

表 3.3　C+ + 中的关系和逻辑运算符

C+ + 运算符	含　义	举　例
&&	逻辑与	if (i > 1 && i < 10)
©©¦¦	逻辑或	if (c= = 0 ©©¦¦c= = 9)
!	逻辑非	if (! (c> 1 && c< 9))
<	小于	if (i < 0)
< =	小于等于	if (i < = 0)
>	大于	if (j > 10)
> =	大于等于	if (x > = 8.2)
= =	等于	if (c = = '0')
! =	不等于	if (c! = '\n')
? :	条件赋值	k = (i< 1) ? 1 : i;

虽然可以在 C+ + 中用这些宏, 但是熟练的 C+ + 程序员读了你的代码以后可能会持相反的态度。谁说程序设计总是客观的?

警　告　不要用= 运算符作为相等关系运算符。这种常见错误是产生 C+ + 程序的逻辑错误的根源之一。在其他语言中你可能已经习惯于用= 运算符去测试两个数据项是否相等, 但在 C+ + 中, 必须使用= = 运算符。如果在 C+ + 中用了= 运算符会产生什么后果? 会出编译错误吗? 回答是可能会有一个编译警告。除此以外, C+ + 程序可以运行, 当程序运行到测试相等关系的表达式时, 它会试图将= 号右边的操作数的值赋给= 号左边的操作数。自然, 这样的程序在运行时会产生一些古怪的行为, 甚至导致系统挂起!

注　意　C++ 中并不支持预定义的布尔标识符，而是将零作为假(false)、非零作为真(true)。为使程序更清晰，建议你声明全局常量 TRUE 和 FALSE，并分别为其赋值 1 和 0。

注意表 3.3 中最后一个运算符是?: 。这种特殊的运算符支持的是条件表达式。

新　词

条件表达式(conditional expression)
条件表达式是具有两重选择的 if-else 语句的简称(关于 if 语句的更详细内容在第 5 天讲述)。

例如，以下是一个 if-else 语句:

```
if (condition)
   variable = expression1;
else
variable = expression2;
```

与其等价的条件表达式是:

```
variable = (condition)? expression1: expression2;
```

条件表达式测试条件值，如果条件为真，则将 expression1 赋给目标变量；否则将 expression2 赋给目标变量。

3.8　布尔表达式

复杂的条件通常需要用一组关系和逻辑运算符组成。这里举几个例子:

```
x < 0 ©©¦x > 11

(i ! = 0 ©©¦i > 100) && (j ! = i ©©¦j > 0)

x ! = 0 && x ! = 10 && x ! = 100
```

新　词

布尔表达式(Boolean expression)
布尔表达式(也称逻辑表达式)是包括逻辑运算符和/或关系运算符的表达式。

要仔细检查布尔表达式以避免出现总是真或总是假的情况。
例如，表达式(x < 0 && x > 10)总为假，因为没有一个值既是负的且又比 10 大。

不要用= 运算符去测试相等关系。

我们来看一个使用关系和逻辑运算符以及表达式的例子。清单 3.6 中列出了程序 RELOP1.CPP 的源代码。程序提示你输入 3 个整数，然后进行一组测试。程序显示关系和逻辑运算、操作数以及结果。

键　入　　清单 3.6　程序 RELOP1.CPP 的源代码

```
/*
simple C++ program that uses logical expressions
this program uses the conditional expression to display
TRUE or FALSE messages, since C++ does not support the
BOOLEAN data type.
*/

#include <iostream.h>

const MIN_NUM = 30;
const MAX_NUM = 199;
const int TRUE = 1;
const int FALSE = 0;

main()
{
  int i, j, k;
  int flag1, flag2, in_range,
     same_int, xor_flag;

  cout << "Type first integer: ";  cin >> i;
  cout << "Type second integer: "; cin >> j;
  cout << "Type third integer: ";  cin >> k;

  // test for name [MIN_NUM..MAX_NUM]
  flag1 = i >= MIN_NUM;
  flag2 = i <= MAX_NUM;
  in_range = flag1 && flag2;
  cout << "\n" << i << "is in the range
       << MIN_NUM << "to " << MAX_NUM << ": "
       << ((in_range) ? "TRUE": "FALSE");

  // test if two or more entered numbers are equal
  same_int = i == j ¦¦ i == k ¦¦ j == k;
   cout << "\nat least two integers you typed are equal: "
        << ((same_int) ? "TRUE": "FALSE");

  // miscellaneous tests
  cout << "\n" << i << "!= " << j << ": "
       << ((i != j) ? "TRUE": "FALSE");
  cout << "\nNOT (" << i << "<" << j << "): "
       << ((!(i < j)) ? "TRUE": "FALSE");
  cout << "\n" << k << ">" << j << ": "
       << ((i <= j) ? "TRUE": "FALSE");
  cout << "\n" << k << ">" << j << ": "
       << ((k > j) ? "TRUE": "FALSE");
  cout << "\n(" << k << "=" << i << ") AND ("
       << j << "!=" << k << "): "
       << ((k == i && j != k) ? "TRUE": "FALSE");

```

```
51:   // NOTE: C++ does NOT support the logical XOR operator for
52:   // boolean expressions.
53:   // add numeric results of logical tests. Value is in 0..2
54:   xor_flag = (k <= i) + (j >= k);
55:   // if xor_flag is either 0 or 2 (i.e. not = 1), it is
56:   // FALSE therefore interpret 0 or 2 as false.
57:   xor_flag = (xor_flag == 1) ? TRUE : FALSE;
58:   cout << "\n(" << k << "<= " << i << ") XOR ("
59:        << j << ">= " << k << "): "
60:        << ((xor_flag) ? "TRUE": "FALSE");
61:   cout << "\n(" << k << "> " << i << ") AND("
62:        << j << "<= " << k << "): "
63:        << ((k > i && j <= k) ? "TRUE": "FALSE");
64:   cout << "\n\n";
65:   return 0;
66: }
```

以下是清单 3.6 中程序的一个运行范例:

输　出

```
Type first integer : 55
Type second integer : 64
Type third integer : 87

55 is in the range 30 to 199 : TRUE
at least two integers you typed are equal : FALSE
55 != 64 : TRUE
NOT (55 < 64) : FALSE
55 <= 64: TRUE
87 > 64 : TRUE
(87 = 55) AND (64 != 87) : FALSE
(87 <= 55) XOR (64 >= 87) : FALSE
(87 > 55) AND (64 <= 87) : TRUE
```

分　析

清单 3.6 中的程序声明了 4 个全局常量。常量 MIN_NUM 和 MAX_NUM 定义了在逻辑测试中所用的数的范围。常量 TRUE 和 FALSE 代表布尔值。main 函数中声明了数个 int 变量以用于输入以及各种测试。第 21 行到 23 行的语句提示你输入 3 个整数并分别将它们存在变量 i, j 和 k 中。

第 26 到 31 行的语句主要用来测试变量 i 的值是否在 MIN_NUM 和 MAX_NUM 之间。第 26 行的语句测试 i 值是否大于等于常量 MIN_NUM。程序将布尔型的结果赋给变量 flag1。第 27 行的语句测试 i 值是否小于等于常量 MAX_NUM。程序将布尔型结果赋值给变量 flag2。第 28 行的语句对变量 flag1 和 flag2 运用 && 运算符并将布尔型结果赋值给变量 in_range。第 29 到 31 行中的输出语句显示测试内容并根据变量 in_range 的值显示 TRUE 或 FALSE。如果 in_range 的值非零, 那么语句用条件运算符?: 显示串

TRUE, 否则显示串 FALSE。

第 34 到 36 行的语句确定你输入的 3 个整数中是否至少有两个相等。第 34 行语句所用的布尔表达式中运用了= = 关系运算符和©©逻辑运算符。该语句将布尔型结果赋给变量 same- int。第 35 和 36 行的输出语句显示测试内容并显示结果为 TRUE 或 FALSE。输出语句使用了条件运算符根据变量 same- int 的值显示串 TRUE 或 FALSE。

第 39 到 49 行中的语句进行杂项测试, 其中用到输入值并且显示测试内容和结果。你应该能自如地修改这些语句以构成不同的测试。

注　意　第 54 到 60 行的语句进行异或(XOR)测试并显示其结果。程序用了一个简单的编程技巧以实施 XOR 运算。第 54 行语句将子表达式(k < = i)和(j > = k)的布尔值相加。如果两个子表达式都为假, 则结果为 0; 如果仅有一个表达式为真, 则结果为 1; 如果两个表达式都为真则结果为 2。既然 XOR 运算符仅在一个子表达式为真时就为真, 如果 xor- flag 原来的值为 1 则第 57 行语句将 TRUE 赋值给变量 xor- flag, 否则语句将 FALSE 赋值给变量 xor- flag。第 61 到 63 行的语句进行另一杂项操作。

3.9　位操作运算符

C+ + 是适合于系统开发的程序设计语言, 而系统开发需要位操作运算符。

新　词

位操作运算符(bit-manipulating operators)
位操作运算符对字节或字的位求反、设置、查询和移位。

表 3.4 中列出了位操作运算符。注意 C+ + 中用 & 和©符号分别代表按位“与”(AND)和按位“或”(OR)。回忆一下, && 和©©字符分别代表逻辑“与”和逻辑“或”运算符。除了位操作运算符以外, C+ + 还支持位操作赋值运算符, 表 3.5 中已列出。(使用位操作运算符属于按位操作的高级程序设计技巧, 作为初学 C+ + 的程序员, 你在近期可能还用不上这些运算符。)

表 3.4　C+ + 位操作运算符

C+ + 运算符	含　义	举　例
&	按位“与”	i & 128
©¦	按位“或”	j ©¦64
^	按位“异或”	j ^ 12
~	按位“非”	~j
< <	向左移位	i < < 2
> >	向右移位	j > > 3

表 3.5　C+ + 位操作赋值运算符

C+ + 运算符	长格式	举　例
x &= y	x = x & y	i &= 128
x + = y	x = x ©¦ y	j ©¦= 64
x ^ = y	x = x ^ y	k ^ = 15
x < < = y	x = x < < y	j < < = 2
x > > = y	x = x > > y	k > > = 3

我们来看一个进行简单位操作的 C+ + 程序。清单 3.7 中列出了程序 BITS1.CPP 的源代码。程序不需要任何输入，因为它使用内部数据。程序中运用了©¦ &，^ ，> > 和 < < 按位运算符并显示按位操作的结果。

键　入　　清单 3.7　程序 BITS1.CPP 的源代码

```
1: // C+ + program to perform bit manipulations
2:
3: # include < iostream.h>
4:
5: main( )
6: {
7:
8:     int i, j, k;
9:
10:    // assign values to i and j
11:    i = 0xF0;
12:    j = 0x1A;
13:
14:    k = j & i;
15:    cout < < j < < "AND "< < i < < "= "< < k < < '\n';
16:
17:    k = j ©¦ i;
18:    cout < < j < < "OR "< < i < < "= "< < k < < '\n';
19:
20:    k = j ^ 0x1C;
21:    cout < < j < < "XOR "< < 0x1C < < "= "< < k < < '\n';
22:
23:    k = i < < 2;
24:    cout < < i < < "shifted left by 2 bits = "< < k < < '\n';
25:
26:    k = i > > 2;
27:    cout < < i < < "shifted right by 2 bits = "< < k < < '\n'
28:    return 0;
29: }
```

以下是清单 3.7 中程序的一个运行范例：

输　出

```
26 AND 240 =  16
26 OR 240 =  250
26 XOR 28 =  6
240 shifted left by 2 bits =  960
240 shifted right by 2 bits =  60
```

分　析

清单 3.7 中的程序声明了 3 个 int 型的变量: i, j 和 k。第 11 和 12 行的语句将十六进制数分别赋给变量 i 和 j。第 14 行的语句对变量 i 和 j 使用按位“与”运算符并将结果存到变量 k 中。第 15 行的输出语句显示操作数、按位运算符以及结果。第 17 行语句对变量 i 和 j 运用按位或运算符并将结果存到变量 k 中。第 18 行的输出语句显示了操作数、按位运算符以及结果。第 20 行语句对变量 j 和十六进制整数 0x1c 运用按位“异或”运算符。第 21 行的输出语句显示了操作数、按位运算符以及结果。

第 23 行到 27 行的语句对变量 i 运用左移和右移运算符。这些运算符将变量 i 移位 2 位并将结果赋给 k。左移运算符的效用与将变量 i 的值乘以 4 是相同的。类似地, 右移运算符的效用与将变量 i 的值除以 4 是相同的。

3.10　逗号运算符

逗号运算符要求完成第 1 个表达式求值后再求第 2 个表达式的值。这两个表达式在同一 C+ + 语句中!“在同一 C+ + 语句中”的确切含义是什么? 为什么要用这种罕见的运算符? 这是因为逗号运算符在 for 循环中起着很特别很重要的作用。

新　词

循环(loop)　　逗号运算符(comma operator)

循环是一种强有力的语言结构, 它使计算机擅长于执行重复性的任务。用逗号运算符可以建立多个表达式以初始化多个与循环有关的变量。

语　法

逗号运算符

逗号运算符的一般语法是:

```
expression1, expression2
```

示　例:

```
for (i =  0, j =  0; i <  10; i+ + , j+ + )
```

第 6 天将学习 for 循环的详细知识, 现在这个例子仅示意如何运用逗号运算符。

运算符优先级和求值方向

至此既然我们已经熟悉了大多数 C+ + 运算符(另外还有一些用于指针和地址的运算符), 就应了解以下两方面的有关知识: 首先是 C+ + 运算符的优先级, 其次是求值的方向(或次序)。表 3.6 中列出了迄今为止所讲述的 C+ + 运算符的优先级并指出了求值

方向。

表 3.6　C++ 运算符及其优先级

种　类	名　称	符　号	求值方向	优先级
单目				
	后增量	++	从左往右	2
	后减量	--	从左往右	2
	地址	&	从右往左	2
	按位“非”	-	从右往左	2
	类型强制	(type)	从右往左	2
	逻辑“非”	!	从右往左	2
	求反	-	从右往左	2
	加号	+	从右往左	2
	前增量	++	从右往左	2
	前减量	--	从右往左	2
	数据大小	sizeof	从右往左	2
乘法				
	取模	%	从左往右	3
	乘法	*	从左往右	3
	除法	/	从左往右	3
加法				
	加法	+	从左往右	4
	减法	-	从左往右	4
移位				
	左移	<<	从左往右	5
	右移	>>	从左往右	5
关系				
	小于	<	从左往右	6
	小于等于	<=	从左往右	6
	大于	>	从左往右	6
	大于等于	>=	从左往右	6
	等于	==	从左往右	7
	不等于	!=	从左往右	7
按位				
	与	&	从左往右	8
	异或	^	从左往右	9
	或	!	从左往右	10
逻辑				
	与	&&	从左往右	11
	或	©©¦¦	从左往右	12
三目	条件表达式	?:	从右往左	13
赋值				
	算术	=	从右往左	14

续表

种　类	名　称	符　号	求值方向	优先级
		+ =	从右往左	14
		- =	从右往左	14
		* =	从右往左	14
		/ =	从右往左	14
		% =	从右往左	14
移位				
		> > =	从右往左	14
		< < =	从右往左	14
按位				
		&=	从右往左	14
		©¦	从右往左	14
		^ =	从右往左	14
逗号		,	从左往右	15

小　　结

今天这一课讲述了各种 C+ + 运算符并讨论了如何用这些运算符操作数据。你所学的内容主要有以下这些:

- 算术运算符包括+ , - , * , / 和% (取模) 。
- 算术表达式变化复杂多端。最简单的表达式仅含一个数据项(文字、常量或变量),复杂的表达式包括多个运算符、函数、文字、常量以变量。
- 增量和减量运算符以前缀和后缀两种格式出现。C+ + 允许你将这些运算符用于字符、整数甚至浮点类型的变量。
- 用算术赋值运算符可以写出更简短的算术表达式,其中主要操作数同时也是接收表达式的结果的变量。
- sizeof 运算符返回一数据类型或变量的字节数。
- 类型强制允许你对表达式进行强制类型转换。
- 用关系和逻辑运算符可以构造逻辑表达式。C+ + 中并不支持预定义的布尔类型而是将零作为假而非零作为真。
- 布尔表达式用关系和逻辑运算符组合形成复杂的条件,这些表达式使程序可以作复杂的决策。
- 条件表达式提供了双重选择的 if-else 语句的简略格式。
- 位操作运算符进行按位的“与”、“或”、“异或”以及“非”运算。此外,C+ + 还支持 < < 和> > 移位运算符。
- 位操作赋值运算符提供了位操作语句的简短格式。

问　与　答

问: 如果声明一个变量但从不给它赋值,编译器会有什么反应?

答: 编译器会警告说该变量未被引用。

问: 如何用布尔表达式检查变量(如 i)的值是否在某个值域中(例如,由 loVal 和 hiVal 所确定)?

答: 确定变量 i 的值是否在值域中的表达式如下:

```
(i >= lowVal && i <= hiVal)
```

问: 如何用布尔表达式检查变量(如 i)的值是否在某个值域内(例如,由 loVal 和 hiVal 所确定)?

答: 确定变量 i 的值是否在值域内的表达式如下:

```
(i > lowVal && i < hiVal)
```

作　　业

作业中所提供的测验题帮助你巩固所学内容,而练习则就使用所学内容提供一些经验。要弄懂测验和练习的解答后再继续下一天的课程。附录 B 的"答案"中提供了解答。

测　验

1. 以下程序的输出是什么?

```
#include <iostream.h>

main()
{
  int i = 3;
  int j = 5;
  double x = 33.5;
  double y = 10.0;

  cout << 10 + j % i << "\n";
  cout << i * i - 2 * i + 5 << "\n";
  cout << (19 + i + j) / (2 * j + 2) << "\n";
  cout << x / y + y / x << "\n";
  cout << i * x + j * y << "\n";
  return 0;
}
```

2. 以下程序的输出是什么?

```
#include <iostream.h>

main()
{
  int i = 3;
  int j = 5;
```

```
  cout << 10 + j % i++ << "\n";
  cout << --i * i - 2 * i + 5 << "\n";
  cout << (19 + ++i + ++j) / (2 * j + 2) << "\n";
  return 0;
}
```

3. **以下程序的输出是什么?**

```
#include <iostream.h>

main()
{
  int i = 3;
  int j = 5;

  i += j;
  j *= 2;
  cout << 10 + j % i << "\n";
  i -= 2;
  j /= 3;
  cout << i * i - 2 * i + j << "\n";
  return 0;
}
```

4. **以下程序的输出是什么?**

```
#include <iostream.h>

main()
{
  int i = 5;
  int j = 10;
  cout << ((i > j) ? "TRUE": "FALSE" << "\n";
  cout << ((i > 0 && j < 100) ? "TRUE": "FALSE") << "\n";
  cout << ((i > 0 && i < 10) ? "TRUE": "FALSE") << "\n";
  cout << ((i == 5 && i == j) ? "TRUE": "FALSE") << "\n";
  return 0;
}
```

练　习

1. **用条件运算符编写函数** max, **它返回两个整数中较大者。**
2. **用条件运算符编写函数** min, **它返回两个整数中较小者。**
3. **用条件运算符编写函数** abs, **它返回整数的绝对值。**
4. **用条件运算符编写函数** isodd, **如果其整数变元为奇数则返回** 0; **否则返回** 1。

第4天

管 理 I/O

C++ 和它的上一代语言 C 一样,没有在语言的核心部分定义 I/O 操作。C++ 和 C 都依赖 I/O 库提供必需的 I/O 支持。这些库主要是面向诸如 MS-DOS 这样的非图形用户界面(GUI)环境的,它们通常用在 QuickWin 应用程序中,这也正是本书介绍这些 I/O 库的原因。但是,本书的主要目的是讲授如何编写 Windows 程序,因此对这些 I/O 库的讨论将尽可能地简短,今天将对 STDIO.H 和 IOSTREAM.H 头文件中所支持的部分输入输出操作和函数进行扼要的介绍,包括以下专题:

- 格式化流输出
- 流输入
- printf 函数

4.1 格式化流输出

C++ 把格式化流的输出作为可扩展 I/O 库中的一组函数。语言设计者们承认,在处理类(类将在第 11 天讲述)时,从 C 语言中继承过来的 STDIO.H 中的 I/O 函数有一定的局限性,因此,在 C++ 中扩展了流的概念。回想一下,C 语言中本来就有流,它是从计算机中某一部分流向另一部分的数据序列。从到目前为止所给出的程序中可以看到,在标准输出流 cout 中使用的是提取运算符<<,也可以看到标准输入流 cin 和插入运算符>>。这一节将介绍有助于进行格式化输出的流函数 width 和 precision。C++ 流库中有许多函数可以微调输出格式。但是,正如前面所述,由于这些函数用于非图形用户界面中,所以这里无意让你沉溺于这些与 Windows 编程不密切的内容中。width 函数说明了输出的宽度。在 cout 流中使用此函数的一般格式为:

```
cout.width(widthOfOutput);
```

precision 函数说明了浮点数位数。在 cout 流中使用此函数的一般格式为:

```
cout.precision(numberOfDigits);
```

我们来看一个例子。清单 4.1 中列出了程序 OUT1.CPP 的源代码。此程序不需要输入,它用 width 和 precision 流函数显示了格式化的整数、浮点数和字符。

键　入　　**清单 4.1　程序 OUT1.CPP 的源代码**

```
1: // Program that illustrates C++ formatted stream output
2: // using the width and precision functions
3:
4: #include <iostream.h>
5:
6: main()
7: {
8:     short aShort          = 4;
9:     int anInt             = 67;
10:    unsigned char aByte   = 128;
11:    char aChar            = '@';
12:    float aSingle         = 355.0;
13:    double aDouble        = 1.130e+002;
14:    // display sample expressions
15:    cout.width(3); cout << int(aByte) << " + ";
16:    cout.width(2); cout << anInt << " = ";
17:    cout.width(3); cout << (aByte + anInt) << '\n';
18:
19:    cout.precision(4); cout << aSingle << " / ";
20:    cout.precision(5); cout << aDouble << " = ";
21:    cout.precision(5); cout << (aSingle / aDouble) << '\n';
22:
23:    cout << "The character in variable aChar is "
24:         << aChar << '\n';
25:    return 0;
26: }
```

下面是清单 4.1 中程序的一个运行范例:

输　出

```
128 + 67 = 195
355 / 113 = 3.1416
The character in variable aChar is @
```

分　析

清单 4.1 中的程序声明了一组具有不同数据类型的变量。第 15 到 17 行语句使用了流函数 width 指定 cout 语句要显示的下一项的输出宽度。注意,为了显示 3 个整数共用了 6 条语句。另外,要注意第 15 行程序使用了表达式 int(aByte) 把 unsigned char 类型强制转换成 int 类型。不做这个类型转换,变量 aByte 的内容将显示为字符而不是整数。如果想用流输出以缺省宽度显示数值,那么可以用一条语句代替上面的 6 条流输出语句。

第 19 到 21 行包含了显示浮点数的第二组流输出语句。这些语句使用了流函数 precision 指定所显示数的总位数。同样,为输出 3 个浮点数,也用了 6 条流输出语句。而

如果用流输出语句以缺省宽度显示数值,也可以用 1 条语句代替这 6 条流输出语句。

4.2　流　输　入

像标准输出流一样,C++ 还提供了标准输入流 cin。该输入流能读入预定义的多种类型的数据,诸如 int, unsigned, long 和 char。一般使用插入运算符>> 获取预定义数据类型的输入。到目前为止所给出的程序中都使用>> 运算符输入一项,而 C++ 流允许用一连串>> 运算符进行多项输入。在多项输入时,需要遵从以下规则:

1. 在连续两个数值之间用空格分隔。
2. 在连续两个 char 之间可任选地加入空格。
3. 仅当 char 以数字形式表达时,char 与数值之间(反之亦然)才须加入空格。
4. 输入流将略去空格。
5. 可以在不同行上输入多项。流输入语句一直到获得所有指定输入时才执行完毕。

> **注　意**　本书对字符串输入的叙述将延后。第 9 天讨论串和串的输入。

我们来看一个包括了多项输入以及不同的数据类型组合的程序示例。清单 4.2 中列出了 IN1.CPP 的源代码。该程序进行以下操作:

- 提示你输入 3 个数;
- 计算这 3 个数的总和;
- 显示所输入的 3 个数的总和及平均值;
- 提示你键入 3 个字符;
- 显示输入内容;
- 提示你输入一个数、一个字符以及另一个数;
- 提示你输入一个字符、一个数字以及另一字符;
- 显示输入内容。

键　入　　**清单 4.2　程序 IN1.CPP 的源代码**

```
1: // Program that illustrartes standard stream input
2:
3: # include < iostream.h>
4:
5: main()
6: {
7:   double x, y, z, sum;
8:   char c1, c2, c3;
9:
10:  cout < < "Enter three numbers separated by a space : ";
11:  cin > > x > > y > > z;
12:  sum = x + y + z ;
```

```
13:   cout << "Sum of numbers = " << sum
14:        << "\nAverage of numbers = " << sum / 2 << "\n";
15:   cout << "Enter three characters: ";
16:   cin >> c1 >> c2 >> c3;
17:   cout << "You entered characters '" << c1
18:        << "', '" << c2 << "', and '"
19:        << c3 << "'\n";
20:   cout << "Enter a number, a character, and a number: ";
21:   cin >> x >> c1 >> y;
22:   cout << "You entered " << x << "'" << c1 << "'" << y << "\n";
23:   cout << "Enter a character, a number, and a character: ";
24:   cin >> c1 >> x >> c2;
25:   cout << "You entered " << c1 << "'" << x << "'" << c2 << "\n";
26:
27:   return 0;
28: }
```

下面是清单 4.2 中程序的一个运行范例:

输　出

```
Enter three numbers separated by a space: 1 2 3
Sum of numbers = 6
Average of numbers = 3
Enter three characters: ABC
You entered characters 'A', 'B', and 'C'
Enter a number, a character, and a number: 12A34.4
You entered 12 A 34.4
Enter a character, a number, and a character: A3.14Z
You entered A 3.14 Z
```

分　析

清单 4.2 中的程序声明了 4 个 double 型变量和 3 个 char 型变量。第 10 行的输出语句提示你输入 3 个数。第 11 行的输入语句读取输入,并把这些数存储在变量 x, y 和 z 中。输入时在任意两个数之间要用空格字符隔开,也可以一个输入数单独占一行。该语句把输入的第一个数存放在变量 x 中;第二个数字存放在变量 y 中;第三个存放在变量 z 中。这个顺序是由这些变量在第 11 行语句中出现的顺序决定的。第 12 行语句计算变量 x, y 和 z 的总和。第 13 和 14 行的输出语句显示了输入数字的总和及平均值。

第 15 行输入语句提示你输入 3 个字符。第 16 行的输入语句读取输入的字符,并依次存储在变量 c1, c2 和 c3 中。输入时字符之间不必用空格隔开。这样,可以用诸如 1A2, Bob 或 1Dd 的方式输入字符。第 17 到 19 行的输出语句显示键入的字符,并用空格分开。

第 20 行的输出语句提示你输入一个数、一个字符以及另一个数。第 21 行的输入语句把输入内容依次存放到变量 x, c1 和 y 中。仅当字符可能被当作数值的一部分时,它们之间才需用空格隔开。例如,如果要输入数 12、圆点字符和数字 55,可键入 12 . 55。圆点前

后的空格可保证输入流不会将它当做浮点数的小数点。第 22 行的输出语句显示了输入值,并且用空格分开。

第 23 行的输出语句提示你输入一个字符、一个数和另一个字符。第 24 行的输入语句把输入内容依次存放到变量 c1, x 和 c2 中。仅当输入的字符有可能被当作数值的一部分时,才需要在字符和空格之间用空格分隔。例如,如果要输入字符- 、数 12 和数字 0,键入- 12 0。第 25 行的输出语句显示了输入值,并且用空格分开。

4.3　printf 函数

对于刚入门的 C++ 程序员来说,可供选用的 I/O 函数非常丰富。这一节将讨论 C 语言标准 I/O 中的 printf 函数的格式化特性,该函数原型在头文件 STDIO.H 中。

printf 函数提供了大量强有力的格式化控制。一条格式指令的一般语法为:

% [flags] [width] [.precision] [F ©¦N ©¦h ©¦l] 〈type character〉

flags 任选项指明了输出对齐方式、数值符号、小数点和尾部零。此外,这些标志还可指定八进制和十六进制的前缀。表 4.1 给出了 printf 函数格式串中标志的任选项。

表 4.1　转 义 序 列

序　列	十进制值	十六进制值	含　义
\a	7	0x07	响铃
\b	8	0x08	退格
\f	12	0x0C	换页
\n	10	0x0A	换行
\r	13	0x0D	回车
\t	9	0x09	水平制表
\v	11	0x0B	垂直制表
\\	92	0x5C	反斜杠
\'	44	0x2C	单引号
\"	34	0x22	双引号
\?	63	0x3F	问号
\000			表示八进制值的 1 到 3 位数
\Xhhh 和 \xhhh		0xhhh	十六进制值

width 任选项指明了最少显示字符数。如果需要,printf 使用零和空格填充。如果宽度数以 0 开始,printf 函数使用前导零,而不用空格进行填充。当用* 字符取代宽度值时,printf 函数将从函数的变元表中获取真实的宽度值。指定所需宽度的变元必须出现在实际要格式化的变元之前。下面的例子用两个字符的宽度显示整数 3,printf 的第 2 个变元指定了宽度 3:

```
printf ( "%* d", 3, 2);
```

precision 任选项指明最多显示字符数。如果包含整数,精度任选项定义最少显示数字位数。当用* 字符代替精度数时,printf 函数从变元表中获取真实的精度值。表明所需精度的参数必须出现在实际要格式化的变元之前。下面的例子以 10 个字符的宽度显示浮点数 3.3244 的小数部分,printf 的第 2 个参数指定精度:

```
printf( %7.*f", 10, 3.3244);
```

F, N, h 和 l 任选项用于取消变元的缺省存储长度而另设长度。F 和 N 分别用与远指针和近指针结合使用,h 和 l 则分别用于表示 short int 或 long。

printf 函数要求每一个% 格式码后都要有指明数据类型的字符。表 4.2 中列出了 printf 格式串中标志的任选项,表 4.3 则给出了 printf 格式串用到的数据类型字符。

表 4.2　printf 函数格式串中的标志任选项。

格式任选项	输出结果
-	指定域左对齐
+	显示数值的正、负
空格	如果值为正,开头显示空格;如果值为负,开头显示负号
#	对十进制整数无影响;对于十六进制整数,开头显示 0X 或 0x; 对于八进制整数,开头显示 0;对于实数,显示小数点

表 4.3　printf 格式串中使用的数据类型字符

类别	类型字符	输出结果
字符	c	单个字符
	d	十进制有符号整数
	i	十进制有符号整数
	o	八进制无符号整数
	u	十进制无符号整数
	x	十六进制无符号整数(所用的数值字符集为 012345689abcdef)
	X	十六进制无符号整数(所用的数值字符集为 0123456789abcdef)
指针	p	对于近指针,只显示偏移地址,格式为 OOOO;对于远指针,显示格式为 SSSS: OOOO
指向整数的指针	n	无输出,printf 语句到目前为止成功写到缓冲区或流中的字符数可以保存到一个变量中,该变量的地址以变元形式给出。
实数	f	以[-]dddd.dddd 格式显示有符号值
	e	以[-]d.ddde[+ ©¦]ddd 格式显示科学计数值
	E	以[-]d.ddddE[+ ©¦]ddd 格式显示科学计数值
	g	根据数值和指定的精度,以 f 或 e 方式显示有符号值
	G	根据数值和指定的精度,以 f 或 E 方式显示有符号值
串指针	s	显示串中空终结符之前的所有字符

> **注　意**　虽然函数 printf 在 Windows 应用程序输出中不起作用，但它的姊妹函数 sprintf 却很有用，后者可以生成包含格式化输出映像的字符串。第 9 天将讨论 sprintf 函数，并在本书以后的课程中使用此函数产生包含有数值信息的对话框。

我们来看一个简单的例子。清单 4.3 中列出了程序 OUT2.CPP 的源代码。该程序是经修改清单 4.1 中的程序 OUT1.CPP 而得到的。新版本使用了 printf 函数来显示格式化输出。程序使用了三组不同的格式码显示同一个浮点数。

键　入　　**清单 4.3　程序 OUT2.CPP 的源代码**

```
1: // C++ program that uses the printf function for formatted output
2:
3: #include <stdio.h>
4:
5: main()
6: {
7: short aShort = 4;
8: int anInt = 67;
9: unsigned char aByte = 128;
10: char aChar = '@';
11: float aSingle = 355.0;
12: double aDouble = 1.130e+002;
13: // display sample expressions
14: printf("%3d %c %2d = %3d\n",
15:     aByte, '+', anInt, aByte + anInt);
16:
17:  printf("Output uses the %%lf format\n");
18:  printf("%6.4f / %6.4lf = %7.5lf\n", aSingle, aDouble,
19:                                  aSingle / aDouble);
20:  printf("Output uses the %%le format\n");
21:  printf("%6.4e / %6.4le = %7.5le\n", aSingle, aDouble,
22:                                  aSingle / aDouble);
23:  printf("Output uses the %%lg format\n");
24:  printf("%6.4g / %7.5lg = %7.5lg\n", aSingle, aDouble,
25:                                  aSingle / aDouble);
26:
27:  printf("The character in variable aChar is %c\n", aChar);
28:  printf("The Ascii code of %c is %d\n", aChar, aChar);
29:  return 0;
30:  }
```

接下来给出是清单 4.3 中程序的一个运行范例。

输　出

```
128 +  67 =  195
Output uses the %lf format
355.0000 / 113.0000 =  3.14159
Output uses the %le format
3.5500e+002 / 1.1300e+002 =  3.141593+000
Output uses the %lg format
   355 / 113 =  3.1416
The character in variable aChar is @
The ASCII code of @ is 64
```

分　析

清单 4.3 中的程序声明了一组具有不同数据类型的变量。第 14 和 15 行的输出语句用%d 和%c 格式控制来显示整数和字符。表 4.4 中列出了第 14 行的 printf 语句中不同格式控制的效果。注意, printf 函数把输出的第一项由 unsigned char 型转换成 int 类型。

表 4.4　第 16 行的 printf 语句中不同格式控制的效果

格式控制	项	数据类型	输出
%3d	aByte	unsigned char	整数
%c	'+'	char	字符
%2d	anInt	int	整数
%3d	aByte + anInt	int	整数

第 18 行的输出语句使用格式控制%6.4f, %6.4lf 和%7.5lf 来显示变量 aSingle, aDouble 和表达式 aSingle / aDouble。这些控制指定的精度分别是 4 位、4 位和 5 位数字, 最小宽度分别为 6 个、6 个和 7 个字符。最后两个格式控制表示显示一个 double 类型的值。

第 21 行的输出语句与第 18 行类似, 主要不同之处在于第 21 行的 printf 函数使用 e 格式代替了 f 格式, 因此, printf 语句中的三项输出均以科学记数法的形式出现。

第 24 行的输出语句也与第 18 行类似, 主要不同之处在于第 24 行的 printf 用 g 格式代替了 f 格式, 因而, printf 语句中的前两项显示时没有小数部分, 因为它们全是整数。

第 27 行的输出语句用%c 格式控制显示了变量 aChar 的内容。第 28 行的输出语句两次显示了变量 aChar 的内容: 一次以字符形式; 另一次以整数(更确切地说, 是字符的 ASCII 码)形式。第 28 行的 printf 函数分别用了%c 和%d 格式控制来完成这一操作。

小　结

今天这一课学习了由 IOSTREAM.H 与 STDIO.H 头文件所支持的基本输入输出操

作以及函数。主要有以下内容:

- 格式化流输出使用 precision 和 width 函数提供一些基本的格式化输出;
- 标准流输入支持用插入运算符获得 C++ 中预定义数据类型的输出;
- printf 函数格式串中所包含的格式代码使得 printf 函数能够控制输出的形式, 甚至可以进行类型转换。

问　与　答

问: 如何将>> 或<< 运算符串在一起?

答: 这些运算符的每一个都返回一个特殊的流数据类型, 它可以用作另一个相同流运算符的输入。

问: 为什么不能在 Windows 应用程序中使用流 I/O 运算符?

答: Windows 应用程序在与用户的交互上有根本的不同之处。当 QuickWin 程序(与非 GUI 的 MS-DOS 应用程序相近) 执行输入语句时, 将进入一种特殊的方式监视键盘的输入。相反, Windows 程序(GUI 应用程序) 则总在监视鼠标(包括它的移动与按钮状态) 和键盘, 并不断把当前状态报告给监视事件的 Windows 部分。GUI 与非 GUI 应用程序的很大的不同之处在于非 GUI 输入函数在 GUI 应用程序中毫无用处。

作　　业

作业部分所提供的测验题有助于巩固理解所学内容, 而通过练习则可以积累运用所学知识的经验。要尽量弄懂测验和练习的答案再继续下一天的课程。附录 B 的"答案"中提供了解答。

测　验

1. 下面的语句有什么错误?

```
cout << "Enter a number" >> x.
```

2. 执行下面的语句会出现什么情况?

```
cout << "Enter three numbers: ";
cin >> x >> y >> x;
```

练　习

1. 编写一个显示从 2 到 10 之间所有数的平方根的程序 OUT3.CPP。利用 MATH.H 头文件中提供的 sqrt 函数计算 double 类型变元的平方根。因为还未讨论C++ 循环, 可利用重复语句显示不同的值。分别用格式控制%3.0lf 和%3.4lf 显示数值和它的平方根。
2. 编写程序 OUT4.CPP, 提示用户输入一个整数, 并以十六进制和八进制形式显示出来。利用 printf 的格式控制进行十进制、十六进制和八进制之间的转换。

第5天

决策机构

对不同的程序设计语言的支持依据其决策机构的变化而变化。

新　词

决策机构(decision-making constructs)
决策机构使应用程序可以进行条件检测并标明动作方向。

今天的课程讨论 C++ 的决策机构,包括下列主题:

- 有一个选择的 if 语句
- 有两个选择的 if-else 语句
- 多个选择的 if-else 语句
- 多个选择的 switch 语句
- 嵌套的决策机构

5.1　有一个选择的 if 语句

与许多程序设计语言不同,C++ 中 if 语句的任何形式都不支持关键词 then。这一语言特性会使你产生 if 语句是如何分隔测试条件与可执行语句的疑问。答案是 C++ 指定用户用圆括号把测试条件括起来。

新　词

有一个选择的 if 语句(single-alternative if statement)
一条 if 语句是有一个选择的语句。

语　法

有一个选择的 if 语句
有一个选择的 if 语句的一般语法格式为:

```
if (condition)
    statement;
```

这是有一条可执行语句的情况。对于有一组执行语句的情况,语法格式一般为:

```
if (tested-condition) {
            < sequence of statements>
}
```

示　例:

```
if (numberOfLines < 0)
   numberOfLines = 0;

if (height - 54) < 3) {
   area = length * width;
```

```
    volume = area * height;
}
```

C++ 使用左右花括号({ })定义语句块。图 5.1 给出了有一个选择的 if 语句流程。

图 5.1　有一个选择的 if 语句的程序流程

我们来看一个例子,清单 5.1 中列出了带有一个选择的 if 语句的程序。该程序它提示你输入一个非零数字并存放在变量 x 中。如果 x 的值非零,程序显示 x 的倒数。

键　入　　清单 5.1　程序 IF1.CPP 的源代码

```
1: // Program that demonstrates the single-alternative if statement
2:
3: # include < iostream.h>
4:
5: main ( )
6: {
7:   double x;
8:   cout < < "Enter a non- zero number : ";
9:   cin > > x;
10:  if (x ! = 0)
11:  cout < < "The reciprocal of "< < x
12:        < < "is "< < (1/x) < < "\n ";
13:  return 0;
14:  }
```

以下是清单 5.1 中程序的一个运行范例:

输　出

```
Enter a non- zero number : 25
The reciprocal of 25 is 0.04
```

分　析

清单 5.1 中的程序在函数 main 中声明了 double 型变量 x。第 8 行的输出语句提示你

输入一个非零数字;第 9 行的输入语句把输入存放在变量 x 中;第 10 行的 if 语句判断 x 是否非 0。

如果此条件为真,程序执行第 11 和 12 行的输出语句,显示 x 的值和它的倒数 1/x;如果测试出条件为假,程序跳过第 11 和 12 行语句,继续第 13 行中的语句。

5.2　有两个选择分支的 if-else 语句

在有两个选择分支的 if 语句中,else 关键字把用于各选择分支的执行语句分隔开。

新　词

有两个选择的 if-else 语句(dual-alternative if-else statement)

有两个选择分支的 if-else 语句提供了依据测试条件布尔值进行的两个分支动作。

语　法

有两个选择的 if-else 语句

对于每一个分支中只有单一可执行语句的情况,有两个选择的 if-else 语句的语法格式一般为:

```
if (condition)
    statement1;
else
    statement2;
```

对于两个分支中有一组执行语句的情况,语法格式一般为:

```
if (tested_condition) {
    < sequence # 1 of statements> }
else {
    < sequence # 2 of statements>
}
```

示　例:

```
if (moneyInAccount > withdraw > {
  moneyInAccount - = withdraw;
  cout < < "You withdrew $ < < withdraw < < "\n";
  cout < < "Balance is $ < < moneyInAccount < < "\n";
}
else {
  cout < < "Cannot withdraw $ "< < withdraw < < "\n";
  cout < < "Account has $ "< < moneyInAccount < < "\n";
}
```

图 5.2 显示了有两个选择分支的 if-else 语句的程序流程。

我们来看一个使用了有两个选择的 if-else 语句的例子。清单 5.2 中列出了程序 IF2.CPP 的源代码。该程序提示用户输入一个字符,然后判断是否输入了一个英文字母。程序将输入分类为英文字母或非英文字母字符进行输出。

键　入　　清单 5.2　程序 IF2.CPP 的源代码

```
1: // Program that demonstrates the dual- alternative if statement
```

```
 2:
 3: # include < iostream. h>
 4: # include < ctype. h>
 5:
 6: main( )
 7: {
 8:   char c;
 9:   cout < <  "Enter a letter : ";
10:   cin > >  c;
11:   // convert to uppercase
12:   c =  toupper(c);
13:   if (c > =  'A' && c < =  'Z')
14:     cout < <  "You entered a letter\n";
15:   else
16:     cout < <  "Your input was not a letter\n";
17:   return 0;
18:   }
```

图 5.2　有两个选择的 if-else 语句的程序流程

以下是清单 5.2 中程序的一个运行范例:

输　出

```
Enter a character : g
You entered a letter
```

分　析

清单 5.2 中的程序在第 8 行声明了一个 char 类型变量 c。第 9 行中的输出语句提示你输入一个英文字母。第 10 行的输入语句读取输入并存放在变量 c 中。第 12 行语句中通过调用函数 toupper(原型在 CTYPE.H 头文件中)把变量的值转换成大写字母。字母大

小写转换简化了第 13 行 if-else 语句的测试条件。if-else 语句判断变量 c 中的字符是否在 A 到 Z 之间。如果条件为真，程序执行第 14 行中的输出语句，显示表明用户输入了英文字母的信息；与此相反，如果测试条件为假，程序执行第 16 行 else 分支中的语句，显示输出表明用户输入非英文字母的信息。

if 语句的潜在问题

在有两个选择的 if 语句中有一个潜在的问题。这一问题发生在 if 分支中包含了另一个有一个选择的 if 语句时，在这种情况下，编译器认为 else 子句附属于最近的 if 语句(最近的一个 if 语句是 if 或 else 子句中包含的另一个 if 语句——在下一节中将有更多层嵌套)。下面是一个例子：

```
if (i > 0)
  if (i = = 10)
      cout < < "You guessed the magic number";
else
  cout < < "Number is out of range";
```

在这个代码段中，当变量 i 是一个不等于 10 的正数时，程序将显示信息 Number is out of range。编译器将这些语句与下列代码段等同看待：

```
if (i > 0)
   if (i = = 10)
       cout < < "You guessed the magic number";
else
   cout < < "Number is out of range";
```

为避开这个问题，可以将语句体中最近的 if 语句括起来：

```
if (i > 0) {
  if (i = = 10)
      cout < < "You guessed the magic number";
}
else
   cout < < "Number is out of range";
```

5.3　有多个选择的 if-else 语句

C+ + 允许 if-else 语句嵌套以产生多路选择分支的形式，从而使应用程序具有更强大的功能和灵活性。

新　词

多个选择的 if-else 语句(multiple-alternative if-else statement)
多个选择的 if-else 语句是嵌套的 if-else 语句。

语　法

多个选择的 if-else 语句
多个选择的 if-else 语句的语法格式一般为：

```
if (tested- condition1)
    statement1; ©¦< sequence # 1 of statement> }
else if (tested- condition2)
```

```
    statement2; ©¦< sequence # 2 of statement>  }
…
else if (tested- conditionN)
    statementN; ©¦< sequence # N of statement>  }
[else
    statementN+ 1; ©¦< sequence # N+ 1 of statement>  }]
```

示　例:

```
char op;

int opOK = 1;
double x, y, z;
cout << "Enter operand1 operator operand2: ";
cin >> x >> op >> y;
if (op == '+')
    z = x + y;
else if (op == '-')
    z = x - y;
else if (op == '*')
    z = x * y;
else if (op == '/' && y != 0)
    z = x / y;
else
    opOK = 0;
```

多个选择的 if-else **语句执行了一系列级联测试直到下列情况之一发生**:

1. if **分支或** else if **分支的条件中有一个为真, 此时执行与其相对应的语句。**
2. **没有一个测试条件为真时, 程序将执行最后一个** else **分支中的语句(如果有** else **分支的话)。**

图 5.3 **中显示了多重选择的** if-else **语句的程序流程。**

我们来看一个例子, 清单 5.3 **因列出了程序** IF3.CPP **的源代码。程序提示用户输入一个字符并利用多个选择的** if-else **语句判断输入是否为下列情况之一**:

- **大写字母**
- **小写字母**
- **数字**
- **非字母数字字符**

键　入　　**清单** 5.3　IF3.CPP **程序的源代码**

```
1: // Program that demonstrates the multiple-alternative if statement
2:
3: # include < iostream.h>
4:
5: main()
6: {
7:   char c;
8:   cout << "Enter a character : ";
9:   cin >> c;
```

```
10:   if (C > = 'A' && c < = 'Z')
11:     cout < < "You entered an uppercase letter\n";
12:   else if (c > = 'a' && c < = 'z')
13:     cout < < "You entered a lowercase letter\n";
14:   else if (c > = '0' && c < = '9')
15:     cout < < "You entered a digit\n";
16:   else
17:     cout < < "You entered a non-alphanumeric character\n";
18:   return 0;
19: }
```

图 5.3　多个选择的 if-else 语句的程序流程

以下是清单 5.3 中程序的一个运行范例:

输　出

```
Enter a character : !
You entered a non- alphanumeric character
```

分　析

清单 5.3 中的程序在第 7 行声明了一个 char 型的变量 c。第 8 行的输出语句提示用户输入一个字符; 第 9 行中的输入语句读取输入并存放到变量 c 中。多个选择的 if-else 语句测试下列条件:

1. 在第 10 行中，if 语句判定变量 c 中的字符是否在 A 到 Z 之间。如果此条件为真，程序执行第 11 行的输出语句，该这条语句证实用户输入了大写字母，然后程序继续执行第 18 行的语句。
2. 如果第 10 行的条件为假，程序跳到第 12 行中的第一个 else if 分支，判断变量 c 中的内容是否为 a 到 z 之间的字母。如果条件为真，程序执行第 13 行的输出语句，这条语句证实输入了小写字母，然后程序继续执行 18 行的语句。
3. 如果第 12 行的条件为假，程序跳到第 14 行中的第二个 else if 分支，判断变量 c 中的内容是否为数字。如果条件为真，程序执行第 15 行的输出语句，该语句证实输入了数字，然后程序继续执行第 18 行的语句。
4. 如果第 14 行的条件为假，程序跳到 16 行的最终 else 分支并执行第 17 行的输出语句。这条语句将显示输入既非英文字母也非数字的信息。

5.4　switch 语句

switch 语句提供了一种特殊形式的多路选择决策机构，用它可以检测一个整型类表达式的各种值并选择相应的分支操作。

语　法

switch 语句

switch 语句的语法格式一般为：

```
switch (expression) {
        case constant1_1:
[       case constant1_2: ...]
              < one or more statements>
              break;
        case constant2_1:
[       case constant2_2: ...]
              < one or more statements>
              break;
...
        case constantN_1:
[       case constantN_2: ...]
              < one or more statements>
              break;
        default:
              < one or more statements>
}
```

示　例：

```
OK = 1;
switch (op) {
    case '+':
        z = x + y;
        break;
    case '-':
        z = x - y;
        break;
    case '*':
```

```
            z = x * y;
            break;
        case '/':
            if (y != 0)
            z = x / y;
        else
            OK = 0;
            break;
        default:
            OK = 0;
}
```

使用 switch 语句的规则是:

1. switch 要求一个与整数类型相容的值。这个值可以是一个常量、变量、函数调用或表达式。switch 语句不能用于浮点数据类型的值。
2. 每一个 case 标号后的值必须为常量。
3. C++ 不支持 case 标号后为一取值范围。相反,每一个值都必须出现在单一的 case 标号之后。
4. 在一组执行语句后需要使用 break 语句。break 语句使程序继续执行当前 switch 语句后的程序。如果不使用 break 语句,程序继续执行紧接着的 case 标号语句。
5. default 分支为最终分支。
6. 在每一个或一组 case 标号中的语句体不需用左右大括号封闭起来。

> **注　意**　如果有一大串连续的取值范围,在一个 case 标号后不能跟以取值范围的缺陷反而使得有多个选择的 if-else 语句更有吸引力。

图 5.4 显示了多路选择 switch 语句的程序流程。

让我们来看一个使用 switch 语句的例子。清单 5.4 中列出了程序 SWITCH.CPP 的源代码,它是通过修改清单 5.3 得到的。新的程序同样完成区分输入字符的任务,但这次利用了 switch 语句。

键　入　　**清单 5.4　SWITCH1.CPP 程序的源代码**

```
1: // Program that demonstrates the multiple-alternative switch statement
2:
3: # include <iostream.h>
4:
5: main()
6: {
7:   char c;
8:   cout << "Enter a character: ";
9:   cin << c;
10:  switch (c) {
11:  case 'A':
12:  case 'B':
13:  case 'C':
```

```
14:   case 'D':
15:   // other case labels
16:      cout << "You entered an uppercase letter\n";
17:      break;
18:   case 'a':
19:   case 'b':
20:   case 'c':
21:   case 'd':
22:   // other case labels
23:      cout << "You entered a lowercase letter\n";
24:      break;
25:   case '0':
26:   case '1':
27:   case '2':
28:   case '3':
29:   // other case labels
30:      cout << "You entered a digit\n";
31:      break;
32:   default;
33:   cout << "You entered a non-alphanumeric character\n";
34:   }
35:   return 0;
36:   }
```

图 5.4　多路选择 switch 语句的程序流程

以下是清单 5.4 中程序的一个运行范例:

输　出

```
Enter a character : 2
You entered a digit
```

分　析

清单 5.4 中的程序声明了 char 类型变量 c;第 8 行中的输出语句提示用户输入一个字符;第 9 行的语句把输入存在变量 c 中;switch 语句从第 10 行开始。第 11 行到 14 行包含了字母 A 到 D 的 case 标号,为使程序简短,这里我们省略了其余大写字母的 case 标号。如果变量 c 中的字符与第 11 到 14 行中的任何一个值匹配,程序便执行第 16 行中的输出语句,确认输入了大写字母(因为减少了 case 标签的数目,所以仅当输入 A 到 D 之间的字母时程序才执行 16 行中的语句)。第 17 行中的 break 语句导致程序跳出 switch 语句,从第 35 行起继续执行。

如果变量 c 中的字符与第 11 到 14 行中的任何一个 case 标号都不匹配,程序从 18 行继续执行,在那里将遇到另一组 case 标号,并假定这些标号代表了小写字母。正如所看到的那样,为使程序简短而减少了标好的数目。如果变量 c 中的字符与第 18 到 21 行中的任何一个值匹配,程序便执行 23 行中的输出语句,证实输入的是小写字母(因为减少了 case 标号的数目,所以仅当输入 a 到 d 间的字母时程序才执行 23 行中的语句)。第 24 行的 break 语句导致程序跳出 switch 语句,从第 35 行起继续执行。

如果变量 c 中的字符与第 18 到 21 行中的任何一个 case 标号都不匹配,程序从第 25 行起继续执行,在那里将遇到另一组 case 标号,可假定这些标号代表了数字。可以看到为使程序简短也减少了标号数目。如果变量 c 中的字符与 25 到 28 行中的任一个值匹配,程序便执行 30 行中的输出语句,证实输入的是数字。(因为减少了 case 标号的数目,所以仅当输入 0 到 3 之间的数字时才执行 30 行中的语句。)31 行中的 break 语句导致程序跳出 switch 语句,从 35 行起继续执行。

如果变量 C 中的字符与 25 到 28 行中的任一个 case 标号都不匹配,程序跳到 32 行的最终分支,执行 33 行中的输出语句,告诉用户输入的是非字母数字字符。

5.5　嵌套的决策机构

为处理复杂条件经常需要采用嵌套的决策机构,这样就可以利用‘分而治之’的策略处理复杂条件。一般是外层结构用于测试基本的或比较概括的条件,而内层结构用于测试更具体的条件。

我们来看一个例子,清单 5.5 中列出了程序 IF4.CPP 的源代码。程序提示用户输入字符,然后判断输入是大写字母、小写字母或非字母字符,显示把输入分类后的信息。

键　入　　**清单 5.5　程序 IF4.CPP 的源代码**

```
1: //Program that demonstrates the nested if statement
2:
3: #include <iostream.h>
4:
5: main()
6: {
7:   char c;
8:   cout << "Enter a character: ";
9:   cin >> c;
10:  if ((C >= 'A' && c <= 'Z') ©¦©¦(c >= 'a' && c <= 'z'))
11:     if (c >= 'A' && C <= 'Z')
12:        cout << "You entered an uppercase letter\n";
13:     else
14:        cout << "You entered a lowercase letter\n";
15:  else
16:        cout << "You entered a non-alphanumeric character\n";
17:  return 0;
18: }
```

以下是清单 5.4 中程序的一个运行范例:

输　出

```
Enter a character: a
You entered a lowercase letter
```

分　析

清单 5.5 中的程序声明了 char 类型的变量 c。第 8 行中的输出语句提示用户输入一个字符; 第 9 行的语句把输入存在变量 c 中; 程序从第 10 行和 11 行开始使用嵌套的 if-else 语句。外层 if-else 语句判断 c 的内容是否为字母。如果测试条件为真, 程序从 11 行的内层 if-else 语句继续执行; 否则, 从外层 if-else 语句的 else 子句处继续执行并选择第 16 行中的输出语句, 表明输入的字符不是字母。

程序使用内层的 if-else 语句进一步仔细检测外层的 if-else 语句中的条件。第 11 行中的 if-else 语句判断变量 c 中的字符是否为大写字母。如果测试条件为真, 程序执行第 12 行的输出语句; 否则, 程序执行第 14 行的 else 语句。这些输出语句告诉你输入的是大写或小写字母。在执行完内层的 if-else 语句后, 程序跳转到第 17 行, 越过外层的 if-else 语句的末尾。

小　　结

今天这一课介绍了 C+ + 中各种决策机构，包括如下内容：

· 有一个选择的 if-else 语句，格式为：

```
if (tested- condition)
    statement; ©¦{ < sequence # 1 of statements > }
```

· 有两个选择的 if-else 语句，格式为：

```
if (tested- condition)
    statement1; { < sequence # 1 of statements> }
else
    statement2; { < sequence # 2 of statements> }
```

· 多个选择的 if-else 语句，格式为：

```
if (tested- condition1)
    statement1; ©¦{ < sequence # 1 of statement> }
else if (tested- condition2)
    statement2: ©¦{ < sequence # 2 of statement> }……
else if (tested- conditionN)
    statementN; ©¦{ < sequence # N of statement> }
[else
    statementN+ 1; ©¦{ < sequence # N+ 1 of statement> }]
```

· 多个选择的 switch 语句，格式为：

```
switch (caseVar) {
        case constant1- 1:
        case constant1- 2:
        < other case labels>
                < one or more statements>
                break;
        case constant2- 1:
        case constant2- 2:
        < other case labels>
                < one or more statements>
                break;
        ....
        case constantN - 1:
        case constantN - 2:
        < other case labels>
                < one of more statements>
                break;

        default:
                < one or more statements>
                break;

}
```

今天我们学习的内容还有：

- if 语句要求遵守两条规则:
 测试条件必须用圆括号括起来;
 语句块要用左右花括号对括起来。
- 采用‘ 分而治之 ’思想的嵌套决策机构增强了处理复杂条件的能力。外层结构用于测试基本的或比较概括的条件, 内层结构用于更加具体的条件。

问　与　答

问: C+ + 对 if 语句分支中的语句缩进有规则限制吗?

答: 没有。缩进纯粹由程序员自己决定。一般的缩进在 2 到 4 个空格之间。使用缩进可以使程序的可读性更好。以下是分支语没有缩进的 if 语句的情形:

```
if (i >  0)
j =  i *  i;
else
j =  10 -  i;
```

与采用了缩进的 if 语句比较一下可读性:

```
if (i >  0)
 j =  i *  i;
else
 j =  10 -  i;
```

显然, 采用缩进的版本可读性更强。

问: 书写 if-else 语句条件的规则是什么?

答: 有两种考虑方法。第一种想法认为要尽量使用取值多为真、少为假的条件; 第二种想法认为要避免否定表达式(使用关系运算符! = 和布尔运算符! 的表达式)。基于这种想法把下面的 if 语句;

```
if (i ! =  0)
   j =  100 / i;
else
   j =  1;
```

翻译成下述等效的语句形式:

```
if (i = =  0)
   j =  1;
else
   j =  100 \ i;
```

即使变量 i 的值为 0 的可能性很小。

问: 如何处理诸如下述除法中除数变量有可能为 0 的条件

```
if ( i ! =  0 && 1/i >  1)
     j =  i *  i;
```

答: C+ + 不总是求整个表达式的值。当布尔表达式中某一项的值使整个表达式的值

为真或假时就只进行部分求值,而略去其他项的值。在这种情况下,如果变量 i 为 0,运行系统不计算项 1/i > 1 的值,因为项 i != 0 为假,即可使整个表达式的值为假,而不管第二个项为何值。

问: 在多个选择的 if-else 和 switch 语句中必须用 else 或 default 子句吗?

答: 程序员们竭力建议包含这些最终子句以保证多个选择的语句处理了所有条件。

作　　业

作业部分所提供的测验题有助于巩固理解所学内容,而通过练习则可以积累运用所学知识的经验。要尽量弄懂测验和练习的答案再继续下一天的课程。附录 B 的"答案"中提供了解答。

测　验

1. 利用一个选择的 if 语句简化下面嵌套的 if 语句:

```
if (i > 0)
    if (i > 0)
      cout << "i = " << i << "\n";
```

2. 利用有一个选择的 if 语句简化下面的 if 语句:

```
if (i > 0) {
  j = i * i;
  cout << "j = " << j << "\n";
}

if (i < 0) {
  j = 4 * i;
  cout << "j = " << j << "\n";
}

if (i == 0) {
  j = 10 + i
  cout << "j = " << j << "\n";
}
```

3. 判断,下面的 if 语句与 if-else 语句执行相同的任务吗?

```
if (i < 0) {
   i = 10 + i;
   j = i * i;
   cout << "i = " << i << "\n";
   cout << "j = " << j << "\n";
}
if (i >= 0) {
   k = 4 * i + 1;
   cout << "k = " << k << "\n";
}
```

```
if (i < 0) {
    i = 10 - i;
    j = i * i;
    cout << "i = " << i << "\n";
    cout << "j = " << j << "\n";
}
else {
    k = 4 * i + 1;
    cout << "k = " << k << "\n";
}
```

4. **简化下面的** if-else **语句**:

```
if (i > 0 && i < 100)
    j = i * i;
else if (i > 10 && i < 50)
    j = 10 + i;
else if (i >= 100)
    j = i;
else
    j = 1;
```

5. **下面的** if **语句错在哪里**?

```
if (i > (1 + i * i)) {
    j = i * i;
    cout << "i = " << i << "and j = " << j << "\n";
}
```

练　习

1. **编写程序** IFT.CPP **求解二次方程式的根。二次方程式为**:

$$AX^2 + BX + C = 0$$

二次方程式的根为:

$$root1 = (-B + \sqrt{(B^2 - 4AC)}) / (2A)$$
$$root2 = (-B - \sqrt{(B^2 - 4AC)}) / (2A)$$

如果平方根中的项为负数, 则根为复数解; 如果平方根项为 0, **则会有两个相同的根**: -B/(2A)。

2. **编写程序** SWITCH2.CPP, **完成简单的四则运算。程序应提示输入算数和算符并显示输入和结果, 还应包括对错误运算符和除数为零的情况的出错检查。**

第6天

循　　环

回顾一下，第 3 天中曾谈及的“循环”是一种强有力的语言机构，有了它，计算机才擅长于完成重复性任务。确切地说，计算机能够准确地、不知疲倦地、快速地重复完成任务，正是它胜过人类的一个方面。今天的课程讲述 C++ 中的下述循环结构：

- for 循环语句
- do-while 循环语句
- while 循环语句
- 跳过循环迭代
- 退出循环
- 嵌套循环

6.1　for　循　环

C++ 中的 for 循环语句是一种多用途的循环，它既支持恒定迭代又支持有条件迭代。for 循环语句的后一特性超越了其他程序设计语言，如 Pascal 和 Basic 中 for 循环的典型用法。

语　法

for 循环

for 循环语句的语法格式一般为：

```
for ( < initialization of loop control variables> ;
      < loop continuation test> ;
      < increment/decrement of loop control variables> )
```

示　例：

```
for (i = 0; i < 10; i++)
    cout << "The cube of " << i << " = " << i * i * i << "\n";
```

for 循环语句分为三个部分，每一部分都是任选的。第一部分用于初始化循环控制变量(C++ 允许使用多个循环控制变量)；循环语句的第二部分是决定循环是否再次迭代的条件；最后一部分是增或减循环控制变量的子句。

注　意　C++ 的循环允许在循环语句中说明循环控制变量。

我们来看一个例子。清单 6.1 中列出了程序 FOR1.CPP 的源代码。程序提示用户通过指定上、下界来定义一个整数范围，然后计算这个范围内整数的和与平均值。

键　入　　**清单 6.1　程序 FOR1.CPP 的源代码**

```
1: // Program that calculates a sum and average of range of
2: // integers using a for loop
3:
4: # include < iostream.h>
5:
6: main ()
7:
8:   double sum =  0;
9:   duuble sumx =  0.0;
10:  int first, last, temp;
11:
12:  cout < <  "Enter the first integer :  ";
13:  cin > >  first;
14:  cout < <  "Enter the last integer :  ";
15:  cin > >  last;
16:  if (first >  last) {
17:     temp =  first;
18:     first =  last;
19:     last =  temp;
20:  }
21:  for (int i =  first; i < =  last; i+ + ) {
22:     sum+ + ;
23:     sumx + =  (double)i;
24:  }
25:  cout < <  "Sum of integers from  "
26:        < <  first < <  "to "< <  last < <  "=  "
27:        < <  sumx < <  "\n";
28:  cout < <  "Average value =  "< <  sumx / sum;
29:  return 0;
30: }
```

以下是清单 6.1 中程序的一个运行范例:

输　出

```
Enter the first integer : 1
Enter the last integer : 100
Sum of integers form 1 to 100 =  5050
Average value =  50.5
```

分　析

清单 6.1 中的程序在函数 main 中声明了一组 int 型以及 double 型的变量。函数把求和变量 sum 和 sumx 初始化为 0。第 12 到 15 行的输入与输出语句提示输入定义取值范围的整数, 并把它们存放在变量 first 与 last 中。第 16 行的 if 语句判断变量 first 中的值是

否大于变量 last 的值。如果条件为真,程序执行第 17 到 19 行中的语句块。这些语句利用变量 temp 作为交换缓冲区,交换变量 first 与 last 的值,这样,if 语句便可保证 first 中的整数小于或等于变量 last 中的整数。

程序通过第 21 行中的 for 循环进行求和。该循环声明了自己的控制变量 i,并且用变量 first 的值对其初始化。循环继续条件是 i < = last,这一条件表明只要 i 小于或等于变量 last 的值,循环便迭代。循环增量部分为 i+ + ,即每迭代一次循环控制变量增加 1。循环中包含两条语句,第一条语句增加变量 sum 的值;第二条语句把变量 sumx 的值增加 i (把类型强制转换为 double)。

注　意　可以改写上面的循环,把循环中的第一条语句移到循环增量部分:

```
for (int i = first; i < = last; i+ + , sum+ + )
    sumx = + (double)i;
```

第 25 到 27 行的输出语句显示指定取值范围中整数的总和与平均值。

为了说明 for 循环语句的灵活性,通过修改程序 FOR1. CPP,建立清单 6. 2 所示的程序 FOR2. CPP。两个程序进行相同迭代,完成相同的任务。不同之处在第 10 行与第 21 到 25 行的语句。第 10 行中声明循环控制变量;第 21 行中用变量 first 的值初始化变量 i;for 循环位于第 22 行。因为第 21 行中已进行了初始化,所以循环没有初始化部分。另外,移去了循环增量部分并通过第 24 行中对变量 i 运用后增量运算符进行补偿。

键　入　　**清单 6. 2　程序 FOR2. CPP 的源代码**

```
1: // Program that calculates a sum and average of range of
2: // integers using a for loop
3:
4: # include < iostream. h>
5:
6: main ( )
7:
8:   double sum = 0;
9:   duuble sumx = 0. 0;
10:  int first, last, temp, i;
11:
12:  cout < < "Enter the first integer : ";
13:  cin > > first;
14:  cout < < "Enter the last integer : ";
15:  cin > > last;
16:  if (first > last) {
17:     temp = first;
18:     first = last;
19:     last = temp;
20:  }
21:  i = first;
22:  for (; i < = last; ) {
```

```
23:     sum+ + ;
24:     sumx + = (double)i+ + ;
25:   }
26:   cout < <  "Sum of integers from "
27:         < <  first < <  "to "< <  last < <  "= "
28:         < <  sumx < <  "\n";
29:   cout < <  "Average value = "< <  sumx / sum;
30:   return 0;
30: }
```

以下是清单 6.2 中程序的一个运行范例:

输　出

```
Enter the first integer : 10
Enter the last integer : 100
Sum of integers from 10 to 100 = 5005
Average value = 55
```

6.2　使用 for 循环的开循环

在介绍 C+ + 的 for 循环时已说明 for 循环的三部分是任选的。实际上, C+ + 允许这三部分均为空!

新　词

开循环(open loop)
循环的三部分都为空的循环便是开循环。

值得指出的是其他语言, 例如 Ada、Modula-2, 也支持规范的开循环并提供退出这种循环的机制。C+ + 允许用下述两种方法之一退出循环:

1. break 语句导致程序执行当前循环后的语句。在希望退出 for 循环时使用 break 语句将继续执行程序其余部分。
2. exit 函数(在 STDLIB.H 中说明)可以退出程序。如果想停止迭代且退出程序, 使用 exit 函数。

我们来看一个例子。清单 6.3 中列出了程序 FOR3.CPP 的源代码。程序使用开循环重复提示用户输入一个数。程序接收输入并显示输入数以及它的倒数值, 然后询问用户否希望计算另一个数的倒数。如果输入字母 Y 或 y, 程序再次迭代; 否则, 程序结束。如果对后一个提示持续不断地输入 Y 或 y, 程序将一直运行 —— 直到计算机坏掉!

键　入　　清单 6.3　程序 FOR3.CPP 的源代码

```
1: // Program that demonstrates using the
2: // for loop to emulate an infinite loop.
3:
4: # include < iostream.h>
```

```
 5: # include < ctype. h>
 6:
 7: main ( )
 8: {
 9:   char ch;
10:   double x, y;
11:
12:   // for loop with empty parts
13:   for (;;) {
14:        cout < < "\nEnter a number : ";
15:        cin > > x;
16:        // process number if non-zero
17:        if ( x ! = 0) {
18:          y = 1/ x;
19:          cout < < "1/ ( "< < x < < ") = "< < y < < "\n";
20:          cout < < "More calculations? (Y/N) ";
21:          cin > > ch;
22:          ch = toupper(ch);
23:          if (ch ! = 'Y')
24             break;
25:             }
26:             else
27:               // display error message
28:               cout < < "Error: cannot accept 0\";
29:   }
30:   return 0:
31: }
```

以下是清单 6.3 中程序的一个运行范例:

输　出

```
Enter a number : 5
1/(5) = 0.2
More calculations? (Y/N) y

Enter a number : 12
1/(12) = 0.0833333
More calculations? (Y/N) y

Enter a number : 16
1/(16) = 0.0625
More calculations? (Y/N) n
```

分　析

清单 6.3 中的程序声明了 char 型变量 c 以及两个 double 类型的变量: x 和 y。函数 main 在 13 行中使用了 for 循环, 是声略了循环的三个组成部分的开循环; 第 14 行的输出语句提示用户输入一个数; 第 15 行的输入语句读取输入并存放在变量 x 中; 第 17 行的

if-else 语句判断变量是否非零。如果条件为真，程序执行第 18 到 24 行的语句块；否则，程序执行第 28 行的 else 子句中的语句，显示一条错误信息。

第 18 行中的语句把变量 x 的倒数赋值给变量 y；第 19 行的输出语句显示变量 x 和 y 的值；第 20 行中的输出语句询问用户是否还要进行计算，要求输入 Y/N（用大写字母或小写字母）回答；第 21 行的输入语句将输入字符存到变量 c 中；第 22 行中的语句利用函数 toupper（函数的原型在 CTYPE.H 头文件中）将输入转换为大写；第 23 中行中的 if 语句判断变量 c 中的字符是否不是 Y，如果条件为真，程序执行第 24 行的 break 语句，这一语句导致程序退出开循环并从第 30 行起继续运行。

6.3　do-while 循环

C++ 中的 do-while 循环是一个条件循环。do-while 循环至少迭代一次。

新　词

条件循环（conditional loop）
只要条件为真，条件循环便进行迭代。在循环尾部测试条件。

语　法

do-while 循环
do-while 循环语法格式一般为：

```
do {
        sequence of statements
} while (condition);
```

示　例：

以下循环显示 2 到 10 的平方：

```
int i = 2;
do {
    cout << i << "^2 = " << i * i << "\n";
} while (++i < 11);
```

我们来看一个例子，清单 6.4 中列出了用来计算平方根的程序 DOWHILE1.CPP 的源代码。程序完成下述任务：

- 提示用户输入一个数（如果输入一个负数，程序重新提示用户输入）；
- 计算并显示输入数的平方根；
- 询问用户是否希望再输入一个数（如果输入字母 Y 或 y，程序重复执行第 1 步；否则程序结束）。

键　入

清单 6.4　程序 DOWHILE1.CPP 的源代码

```
1: // Program that demonstrates the do-while loop
2:
3: #include <iostream.h>
4:
5: const double TOLERANCE = 1.0e-7;
6:
```

```
 7: double abs(double x)
 8: {
 9:   return (x >= 0) ? x : -x;
10: }
11:
12: double sqroot(double x)
13: {
14:   double guess = x / 2;
15:   do {
16:       guess = (guess + x / guess ) / 2;
17:   } while (abs(guess * guess - x) > TOLERANCE);
18:   return guess;
19: }
20:
21: double getNumber()
22: {
23:   double x;
24:   do {
25:     cout << "Enter a number: ";
26:     cin >> x;
27:   } while (x < 0);
28:   return x;
29: }
30:
31: main ()
32: {
33:     char c;
34:     double x, y;
35:
36:     do {
37:       X = getNumber();
38:       Y = sqroot(x);
39:       cout << "Sqrt(" << x << ") = " << y << "\n"
40:            << "Enter another number? (Y/N) ";
41:       cin >> c;
42:       cout << "\n";
43:   } while (c == 'Y' || c == 'y');
44:   return 0;
45: }
```

下面是清单 6.4 中程序的一个运行范例:

输　出

```
Enter a number: 25
Sqrt(25) = 5
Enter another number? (Y/N) y

Enter a number: 144
Sqrt(144) = 12
Enter another number? (Y/N) n
```

分　析

清单 6.4 中的程序中声明了全局常量 TOLERANCE 以及函数 abs, sqroot, getNumber 和 main。函数 abs 位于第 7 行, 它返回 double 型变元的绝对值。

函数 sqroot 位于第 12 行, 它返回参数 x 的平方根。函数在 14 行中设置平方根的初始猜测值为 x/2, 然后利用 do-while 循环对平方根的猜测值进行迭代逼近。while 子句中的条件判断当前估值的平方与参数 x 的差的绝对值是否大于允许的误差(由常量 TOLERANCE 说明)。只要条件为真, 循环便迭代。第 18 行中返回平方根的估值。函数 sqroot 使用牛顿迭代法得到一个数的平方根。

位于第 21 行的函数 getNumber 提示用户输入一个数并存放在局部变量 x 中。函数使用 do-while 循环保证输入的为非负数。第 27 行的 while 子句判断变量 x 的值是否为负数。只要条件为真, do-while 循环便迭代。第 28 行的 return 语句返回 x 的值。

位于第 31 行的函数 main 利用 do-while 循环完成以下任务:

- 调用函数 getNumber 提示用户输入一数字(第 37 行中的语句用了函数调用并把结果赋值给局部变量 x);
- 调用函数 sqroot 计算 x 的平方根, 并赋值给变量 y(第 38 行中有函数调用语句);
- 显示变量 x 和 y 的值;
- 询问用户是否想输入另一个数字(第 41 行的输入语句读取输入字符 Y/N 并存放在变量 c 中)。

位于第 43 行的 while 子句判断变量 c 中的字符是否为字母 Y 或 y。只要条件为真, do-while 循环便进行迭代。

清单 6.4 中的程序示范了 do-while 循环的如下用法:

1. 迭代法计算。函数 sqroot 中的循环示意了这种用法。
2. 数据合法性验证。函数 getNumber 中的循环示意了这种用法。
3. 程序继续。函数 main 中的循环示意了这种用法。

6.4　while 循　环

while 循环是 C++ 中另一种只要条件为真便进行迭代的条件循环。如果一开始测出条件值为假, while 循环将不迭代。

语　法

while 循环

while 循环语法格式为:

```
while (condition)
      statement; ©¦{ sequence of statements)
```

示　例:

```
function power(double x, int n)
{
double pwr =  1;
```

```
while (n- -  >  0)
       pwr * =  x;
return pwr;
}
```

我们来看一个例子,清单 6.5 中列出了程序 WHILE1.CPP 的源代码。程序完成与清单 6.1 中的程序 FOR1.CPP 相同的操作。两个程序交互界面相同并产生同样的结果。

键　入　　**清单 6.5　程序 WHILE1.CPP 的源代码**

```
 1: // Program that demonstrates the do- while loop
 2:
 3: # include < iostream.h>
 4:
 5: main ()
 6: {
 7:   double sum =  0;
 8:   double sumx =  0.0;
 9:   int first ,last, temp, i;
10:
11:   cout < <  "Enter the first integer :  ";
12:   cin > >  first;
13:   cout < <  "Enter the last integer :  ";
14:   cin > >  last;
15:   if (first >  last;
16:     temp=  first;
17:     first =  last;
18:     last =  temp;
19:   }
20:   i =  first;
21:   while (i < =  last) {
22:     sum+ + ;
23:     sumx +  (double)i+ + ;
24:   }
25:   cout < <  "Sum of integers from  "
26:         < <  first < <  "to  "< <  last < <  "=  "
27:         < <  sumx < <  "\n";
28:   cout < <  "Average value =  "< <  sumx / sum;
29:   return 0;
30: }
```

以下是清单 6.5 中程序的一个运行范例:

输　出

```
Enter the first integer :  1
Enter the last integer :  100
Sum of integers from 1 to 100 =  5050
Average value =  50.5
```

分　析

因为清单 6.3 与 6.1 相似,因此集中分析两个程序的主要不同之处,即第 20 到 24 行。第 20 行的语句把变量 first 的值赋给变量 i, while 循环从第 21 行开始。只要变量 i 的值小于或等于变量 last 的值,循环便迭代。变量 i 起循环控制变量的作用。第 22 行语句增加变量 sum 的值。第 23 行的语句把变量 sumx 的值增加 i,同时也增加变量 i 的值。语句通过后增量运算符完成后一任务。

6.5　跳过循环迭代

C++ 允许使用 continue 语句跳到循环末尾并继续下一次迭代。这一程序特性使得循环可以跳过对可能引起运行错误的特殊值的迭代。

语　法

continue 语句

使用 continue 语句的一般格式为:

```
< loop-start clause>  {
        // sequence # 1 of statements
        if (skipCondition)
            continue;
        // sequence # 2 of statements
} < loop-end clause>
```

示　例:　(在 for 循环中):

```
double x, y;
for (int i = -10; i < 11; i++) {
  x = i;
  if (i == 1)
    continue;
  y = 1/sqrt(x * x - 1);
  cout << "1/sqrt(" << (x* x- 1) << ") = " << y << "\n";
}
```

这一形式表明,for 循环中的第一条赋值语句导致了 if 语句中的条件测试。如果条件为真,if 语句便借用 continue 语句跳过 for 循环中的第二部分语句。

我们连看一个例子。清单 6.6 中列出了程序 FOR4.CPP 的源代码。程序显示函数 $f(X) = \overline{(X^2 - 9)}$ 在 x 为- 10 到 10 之间的整数的取值表。因为- 2 到 2 之间的整数产生复数结果,程序要避免这一点,所以表中不显示 f(x) 在- 2 到 2 之间的复数值。

键　入　　清单 6.6　程序 FOR4.CPP 的源代码

```
1: // Program that demonstrates using the continue statement
2: // to skip iterations.
3:
4: # include < iostream.h>
```

```
 5: # include < math.h>
 6:
 7:
 8: double f(double x)
 9: {
10:   return sqrt(x * x - 9);
11: }
12:
13: main ()
14: {
15:   double x, y;
16:
17:   cout < < "          X";
18:   cout < < "          f(X)\n;
19:   cout < < "-----------------------------------------\n\n";
20:   // for loop with empty parts
21:   for (int i = - 10; i < = 10; i+ + ) {
22:     if (i > - 3 && i < 3)
23:       coutinue;
24:     x = (double)i;
25:     y = f(x);
26:     cout < < "       ";
27:     cout.width(3);
28:     cout < < x < < "       ";
29:     cout.width(7);
30:     cout < < y < < "\n";
31:   }
32;   return 0;
33: }
```

以下是清单 6.6 中程序的一个运行范例:

输　出

```
 X              f(X)
 - 10          9.53939
   - 9         8.48528
   - 8         7.4162
   - 7         6.32456
   - 6         5.19615
   - 5               4
   - 4         2.64575
   - 3               0
     3               0
     4         2.64575
     5               4
     6         5.19615
     7         6.32456
     8          7.4162
     9         8.48528
    10         9.53939
```

分　析

清单 6.6 中的程序声明的函数 f 代表数学函数 f(X)。函数 main 在第 15 行中声明了 double 型的变量 x 和 y。第 17 到 19 行中的输出语句显示表头。第 21 行的 for 循环声明了自己的控制变量并在- 10 到 10 之间以 1 为增量进行迭代。循环中的第一句是位于第 22 行的 if 语句,这条语句判断变量 i 的值是否大于- 3 并且小于 3。如果条件为真,程序执行第 23 行的 continue 语句,这样,if 语句能使 for 循环跳过导致产生错误的迭代并继续下一次迭代。第 24 行中的语句将变量 i 的值赋给变量 x。第 25 行的语句以 x 为参数调用函数 f,然后将结果赋给变量 y。第 25 到 30 行的输出语句显示变量 x 和 y 的值,并使用函数 width 进行简单的格式化。

6.6　退 出 循 环

C+ + 支持用于退出循环的 break 语句。break 语句使程序继续执行当前循环之后的语句。

语　法

break 语句

在循环中使用 break 语句的一般格式为:

```
< start-loop clause>  {
      // sequence # 1 of statements
      if (exitLoopCondition)
         break;
      // sequence # 2 of statements
} < end-loop clause>
// sequence # 3 of statements
```

示　例:

```
// calculate the factorial of n
factorial = 1;
for (int i = 1; ; i+ + ) {
  if (i > n)
     break;
  factorial * = (double)i;
}
```

这一形式表明,for 循环中的第一条赋值语句导致了 if 语句中的条件测试。如果条件为真,if 语句便用 break 语句退出整个循环,程序继续执行第三部分语句。

建议你重新读一下清单 6.3 中的程序 FOR3.CPP,这是运用 break 语句的一个极好的例子。

6.7　嵌 套 循 环

嵌套循环使用户能够把一些重复性的任务作为另一些重性任务的一部分。只要需要,C+ + 中允许在任何层次中嵌套任一种循环。嵌套循环常用在处理数组上(这将在第 7 天

讲述)。

我们来看一个使用嵌套循环的例子,清单 6.7 中列出了程序 NESTFOR1.CPP 的源代码。程序显示 1 到 10 之间所有数字的平方根的取值表。程序利用外层循环在上述数值范围内迭代,利用内层循环迭代计算平方根。

键　入　　**清单 6.7　程序 NESTFOR1.CPP 的源代码**

```
1: // Program that demonstrates nested loops
2:
3: # include < stdio.h>
4:
5: const double TOLERANCE =  1.0e- 7;
6: const int MIN- NUM =  1;
7: const int MAX- NUM =  10;
8:
9: double abs(double x)
10: {
11:   return (x > =  0) ? x: - x;
12: }
13:
14: main( )
15: {
16:   double x, sqrt;
17:
18:   print ( "X   Sqrt( X)\n");
19:   print ( "-----------------------------\n\n");
20:   // outer loop
21:   for (int i =  MIN- NUM; i < =  MAX- NUM; i+ + ) {
22:      x =  (double)i;
23:      sqrt =  x /2;
24:      // inner loop
25:      do {
26:        sqrt =  (sqrt +  x / sqrt) / 2;
27:      } while (abs(sqrt *  sqrt -  x) >  TOLERANCE);
28:      printt( "%4.1f      %8.6lf\n", x, sqrt);
29:   }
30:   return 0;
31: }
```

以下是清单 6.7 中程序的一个运行范例:

输　出

X	Sqrt(X)
1.0	1.000000
2.0	1.414214
3.0	1.732051
4.0	2.000000

```
 5.0      2.236068
 6.0      2.449490
 7.0      2.645751
 8.0      2.828427
 9.0      3.000000
10.0      3.162278
```

分　析

清单 6.7 中的程序包含头文件 STDIO.H，为的是使用具有强大格式化输出功能的 printf 函数。第 5 到 7 行定义了常量 TOLERANCE，MIN_NUM 和 MAX_NUM，分别代表平方根值的公差、输出表格中的第一个数和最后一个数。程序中还定义了返回一 double 型数的绝对值的函数 abs。

函数 main 中声明了 double 型变量 x 和 sqrt。第 18 和 19 行的输出语句显示表头；第 21 行中的外层循环是一个 for 循环，循环说明了它自己的控制变量 i，并以 1 为增量从 MIN_NUM 到 MAX_NUM 进行迭代；第 22 行把变量 i 的值存到变量 x 中，第 23 行中的语句取得平方根的初始估值并存到变量 sqrt 中；第 25 行中的内层循环是一个 do-while 循环，它迭代逼近平方根的估值；第 26 行用估值逼近平方根。第 27 行中的 while 子句判断逼近的估值是否足够精确；第 28 行的输出语句显示变量 xt 和 sqrt 的格式化值。

小　结

今天的课程讲述了 C++ 循环以及一些相关内容，学习了下列内容：

- C++ 中的 for 循环有下述的语法格式：

```
for (< initialization of loop control variables> ;
< loop continuation test> ;
< increment/decrement of loop control variables> )
```

for 循环包含三部分内容：循环初始化、循环继续条件和循环变量的增量或减量；

- 条件循环 do-while 的语法格式如下：

```
do {
     sequence of statements
} while (condition);
```

do-while 循环至少迭代一次；

- 条件 while 循环的语法格式如下：

```
while (condition)
       statement; ©¦{ sequence of statements }
```

如果测试条件初始为假，while 循环将不迭代；

- continue 语句可以跳到循环末尾并进行下一次迭代。continue 语句的优点是不用标号控制跳转；
- 开循环是循环三部分都为空的 for 循环，break 语句可以跳出当前循环并继续执

行紧接在循环后面的语句。exit 函数(在 STDLIB.H 中声明)可以通过中止 C++ 程序的运行而完全退出循环;

· 嵌套循环允许把一些重复性的任务作为其他复复性任务的一部分。如果需要,C++ 允许在任何层次上嵌套任一种循环。

问　与　答

问: 如何用 while 循环模拟 for 循环?

答: 下面是一个简单的例子:

```
for (int i = 1; i <= 10; i += 2)
  cout << i << "\n";
```

```
i = 1;
while (i <= 10) {
  cout << i << "\n";
     i+= 2;
     }
```

while 循环需要一个前导语句初始化循环控制变量,另外,它还要求在循环内部用语句改变循环控制变量。

问: 如何用 while 循环模拟 do-while 循环?

答: 下面是一个简单的例子:

```
i = 1;
do {
  cout << i << "\n";
  i += 2;
} while (i <= 10);
```

```
i = 1;
while (i <= 10) {
  cout << i << "\n";
i += 2;
}
```

两个循环的 while 子句中具有相同的条件。

问: 如何用开 for 循环模拟 while 和 do-while 循环?

答: 可以通过在开 for 循环的开始或结尾放置控制循环退出 if 语句来模拟其它 C++ 循环。下面是开 for 循环如何模拟 while 循环的例子:

```
i = 1;
while ( <= 10) {

  cout << i << "\n"
  i += 2;
}
```

```
i = 1;
for (;;) {
  if(i > 10) break;
  cout << i << "\n"
  i += 2;
}
```

注意在开 for 循环的循环内的第一条语句便是控制循环退出的 if 语句。if 语句的测试条件是 while 循环条件的逻辑反。下面例子示意对 do-while 循环的模拟:

```
i = 1;
do {
  cout << i << "\n"
  i += 2;

} while (i <= 10);
```

```
i = 1;
for (;;) {
  cout << i << "\n"
  i += 2;
  if (i > 10) break;
}
```

开 for 循环在循环结尾之前使用了控制循环退出的 if 语句。if 语句的测试条件是 do-while 循环的逻辑反。

问: 在嵌套循环中, 可以把外层循环控制变量作为内层循环取值范围的一部分吗?

答: 可以, C++ 不反对这种用法。下面是一个简单的例子:

```
for (int i = 1; i <= 100; i += 5)
    for (int j = i; j <= 100; j++)
        cout << i * j << "\n";
```

问: C++ 限制嵌套的循环类型吗?

答: 不限制, 在 C++ 程序中可以进行任意循环组合嵌套。

作　　业

作业部分所提供的测验题有助于巩固理解所学内容, 通过练习可以积累运用所学知识的经验。要尽量弄懂测验和练习的答案再继续下一天的课程。附录 B 的‘答案’中提供了解答。

测　验

1. 下面的循环中有何错误?

```
i = 1;
while (i < 10) {
j = i * i - 1;
k = 2 * j - 1;
cout << "i = " << i << "\n";
cout << "j = " << j << "\n";
cout << "k = " << k << "\n";
```

2. 以下循环的输出是什么?

```
for (int i = 5 ; i < 10; i + 2 )
    cout << i - 2 << "\n";
```

3. 以下循环的输出是什么?

```
for (int i = 5; i < 10;)
    cout << i << "\n";
```

4. 下面的代码中有何错误?

```
for (int i = 1; i <= 10; i++)
    for (i = 8; i <= 12; i++)
        cout << i << "\n";
```

5. 下面的嵌套循环中有何错误?

```
for (int i = 1 ; i <= 10; i++)
    cout << i * i << "\n";
for (int i = 1; i <= 10; i++)
```

```
cout << i * i * i << "\n";
```

6. 下面的循环中有何错误?

```
i = 1;
while (1 > 0) {
 cout << i << "\n";
 i++;
}
```

7. 一个数的阶乘是从 1 到这个数之间所有整数的乘积。下面的公式定义了阶乘(使用符号“!”表示阶乘)。

```
n! = 1 * 2 * 3 * ... * n
```

下面的 C++ 程序计算一个数的阶乘。问题是无论输入什么正数,程序显示的阶乘值都为 0,问题出在哪里?

```
int n;
double factorial;
cout << "Enter positive integer : ";
cin >> n;
for (int i = 1; i <= n; i++)
  factorial * = i;
cout << n << "! = "<< factorial;
```

练　习

1. 用 for 循环编写程序 FOR5.CPP,求出并显示 11 到 121 之间所有奇数的总和。
2. 用 while 循环编写程序 WHILE2.CPP,求出并显示 11 到 121 之间所有奇数的平方的总和。
3. 用 do-while 循环编写程序 DOWHILE2.CPP,求出并显示 11 到 121 之间所有奇数平方的总和。

第7天

数　组

数组是最常用的数据结构之一。程序可以用数组存储数据留待以后处理。大多数常用的程序设计语言均支持静态数组,许多语言还支持动态数组。

新　词

数组(array)
数组就是变量组。

今天学习如下一些有关静态数组的专题:

- 声明一维数组
- 使用一维数组
- 初始化一维数组
- 声明作为函数参数的一维数组
- 数组排序
- 数组查找
- 声明多维数组
- 使用多维数组
- 初始化多维数组
- 声明作为函数参数的多维数组

7.1　声明一维数组

一维数组是最简单的数组类型。在一维数组中,对每个变量的访问都是通过单一的索引独立地进行的。

新　词

一维数组(single-dimensional array)
一维数组是一组共享同一名字(数组的名字)的变量。

语　法

一维数组
声明一维数组的语法格式一般为:

```
type arrayName[numberOfElements];
```

在C++中,声明一维数组时需要遵循以下规则:

1. C++数组的下界被置为0,不允许设置或改变下界。
2. 声明C++数组必须说明元素数目。记住,元素数目等于数组上界加1。

在上面的一般格式中,索引的有效范围在0到numberOfElements-1。

示　例:

```
int intArray[10];
char name[31];
double x[100];
```

7.2　使用一维数组

要使用一维数组,就要指明数组的名字和有效的索引值以访问其成员。可以用数组元素存储一个值,也可以再取出来,这取决于对数组元素的引用出现于何处。记住下面的简单规则:

1. 在从数组元素中取出数据之前,必须对它赋值。
2. 使用有效索引。

要对访问数组的索引做合法性检查。
不要假设索引总是合法的。

我们来看一个例子。清单 7.1 中列出了程序 ARRAY1.CPP 的源代码。程序中使用了有 30 个元素的数值数组,计算数值数组中数据的平均值。程序完成以下操作:

- 提示用户输入实际数据点的数目(该值必须在提示信息指明的有效值范围之内);
- 提示输入数组元素的数据;
- 计算数组中数据的平均值;
- 显示平均值。

键　入　　**清单 7.1　程序 ARRAY1.CPP 的源代码**

```
1: /*
2: C++ program that demonstrates the use of one-dimension
3: arrays. The average value of the array is calculated.
4: */
5:
6: #include <iostream.h>
7:
8: const int MAX = 30;
9:
10: main()
11: {
12:
13:     double x[MAX];
14:     double sum, sumx = 0.0, mean;
15:     int i, n;
16:
```

```
17:     do { // obtain number of data points
18:     cout < < "Enter number of data points [2 to "
19:           < < MAX < < "]:";
20:         cin > > n;
21:          cout < < "\n";
22:     } while (n < 2 ©©¦n > MAX);
23:
24:     // prompt user for data
25:     for (i = 0; i < n; i+ + ) {
26:          cout < < "X[" < < i < < "]:";
27:          cin > > x[i];
28:     }
29:
30:     // initialize summations
31:     sum = n;
32:
33:     // calculate sum of observations
34:     for (i = 0; i < n; i+ + )
35:           sumx + = x[i];
36:
37:     mean = sumx / sum; // calculate the mean value
38:     cout < < "\nMean = " < < mean < < "\n\n";
39:     return 0;
40: }
```

以下是清单 7.1 中程序的一个运行范例:

输　出

```
Enter number of data points [2 to 30] : 5

    x[0]: 12.5
    x[1]: 45.7
    x[2]: 25.6
    x[3]: 14.1
    x[4]: 68.4

    Mean = 33.26
```

分　析

清单 7.1 中的程序声明了全程常量 MAX 作为数组大小。函数 main 在第 13 行声明了具有 MAX 个元素的 double 型数组 x。在第 14 和 15 行还声明了其他非数组型变量。

第 17 到 22 行的 do-while 循环读取希望存储在数组 x 中的数据点的数目。第 18 和 19 行的输出语句提示用户输入数据点的数目,输出中指明了有效的数值范围是 2 到 MAX。第 20 行的语句读取输入并存放在变量 n 中。while 子句检查输入合法性,判断变量 n 的值是否小于 2 或大于 MAX,如果此条件为真,do-while 循环再次迭代以读取正确的输入值。

第 25 到 28 行的 for 循环语句提示用户输入数据。循环利用控制变量 i 以 1 为增量从

0 到 n- 1 进行迭代。第 26 行中的输出语句提示输入指定的数组元素的值。第 27 行的输入语句读取输入并存储在元素 x[i] 中。

第 31 行语句把变量 n 中的整数赋值给 double 型的变量 sum。第 34 和 35 行的 for 循环将数组 x 中的值累加到变量 sumx 中。循环利用控制变量 i 以 1 为增量从 0 到 n - 1 进行迭代。第 35 行的语句使用增量赋值运算符将元素 x[i]的值累加到变量 sumx 中。

第 37 行中的语句计算平均值并存入变量 mean;第 38 行中的输出语句显示平均值。

注 意 清单 7.1 中的程序显示了如何用 for 循环处理数组元素。循环使用< 运算符和大于最大索引的值不断进行测试;也可以使用< = 运算符后跟以最大索引值进行测试。例如,输入循环可如下编写:

```
24:     // prompt user for data
25:     for (i = 0; i < = (n - 1); i+ + ) {
26:         cout < < "X[" < < i < < "] : " ;
27:         cin > > x[i];
38:     }
```

但是,因为这一形式比 i < n 中多用一个算符,所以并不通用。

要用最少的运算符编写循环条件表达式,这种方法可减少代码长度并加速循环执行。

不要在循环条件表达式中使用"< = "算符,除非使用该运算符有助于编写减少操作次数的表达式。

7.3 初始化一维数组

在 C+ + 可以对数组进行初始化,并且初始化方法灵活多样。初始值表要用一对匹配的大括号封闭起来,表中元素用逗号分隔并可占多行。如果表中项少于数组元素数目,编译器对余下的元素赋值 0;相反,如果初始值个数多于数组元素数目,编译器产生编译错误。

下一个程序见清单 7.2,它把上一个程序修改为从内部提供数据。因而略去了要求输入数据数目和数据自身的步骤。程序简单显示数组元素(从初始表中获得)和数据的平均值。虽然程序不与用户交互,但它在源代码中存储了数据。在重新计算一个新平均值之前,可以通过编辑程序添加、修改以及删除数据。

键　入　　**清单 7.2　程序 ARRAY2.CPP 的源代码**

```
1: /*
2:    C++ program that demonstrates the use of single- dimensional
3:    arrays. The avergae value of the array is calculated.
4:    The array has its values preassigned internally.
5: */
6:
7: #include <iostream.h>
8:
9: const int MAX = 10;
10:
11: main()
12: {
13:
14:    double x[MAX] = { 12.2, 45.4, 67.2, 12.2, 34.6, 87.4,
15:                      83.6, 12.3, 14.8, 55.5};
16:    double sum = MAX, sumx = 0.0, mean;
17:    int n = MAX;
18:
19:    // calculate sum of observations
20:    cout << "Array is:\n";
21:    for (int i = 0; i < n; i++) {
22:          sumx += x[i];
23:          cout << "x[" << i << "] = " << x[i] << "\n";
24:    }
25:
26:    mean = sumx / sum; // calculate the mean value
27:    cout << "\nMean = " << mean << "\n\n";
28:    return 0;
29: }
```

以下是清单 7.2 中程序的一个运行范例:

输　出

```
Array is:
x[0] = 12.2
x[1] = 45.4
x[2] = 67.2
x[3] = 12.2
x[4] = 34.6
x[5] = 87.4
x[6] = 83.6
x[7] = 12.3
x[8] = 14.8
x[9] = 55.5

Mean = 42.52
```

分　析

首先集中分析清单 7.2 中数组 x 的初始化。第 14 行中声明了数组 x 并对它进行初始化。初始值表一直到第 15 行,它用一对大括号封闭起来并以逗号分隔其值。第 16 行中的语句说明变量 sum 和 sumx,并且这些变量分别初始化为 MAX 和 0。第 17 行中的语句说明 int 类型的变量 n 并用值 MAX 初始化。余下的程序类似于清单 7.1 中的部分。

如果对必须统计初始化值的准确数目感到有些沮丧,那么下面就是好消息:C+ + 可以用相应初始值表中项的数目自动初始化数组大小,因此不必在数组方括号中放入数字,可以让编译器替你干。

要在初始值表中放入虚设值,如果初始化后的数组在以后需要扩展的话。
不要依赖于对初始值表中项的数目计数提供数组元素的数目数据。

清单 7.3 中列出了程序 ARRAY3.CPP 的源代码。这一新版本使用了自动设置数组大小的特性

键　入

清单 7.3　程序 ARRAY3.CPP 的源代码。

```
1: /*
2:   C+ +  program that demonstrates the use of single- dimensional
3:   arrays. The avergae value of the array is calculated.
4:   The array has its values preassigned internally.
5: * /
6:
7: # include < iostream.h>
8:
9: main()
10: {
11:
12:    double x[ ] =  {12.2, 45.4, 67.2, 12.2, 34.6, 87.4,
13:                    83.6, 12.3, 14.8, 55.5};
14:    double sum, sumx =  0.0, mean;
15:    int n;
16:
17:    n =  sizeof(x) / sizeof(x[0]);
18:    sum= n;
19:
20:    // calculate sum of observations
21:    cout < <  "Array is :\n";
22:    for (int i =  0; i <  n; i+ + ) {
23:         sumx + =  x[i];
24:         cout < <  "x[" < <  i < < "]= " < <  x[i] < < "\n";
25:    }
26:
27:    mean =  sumx/sum; // calculate the mean value
```

```
28:    cout < < "\nNumber of data points = " < < n < < "\n"
29:         < < "Mean = " < < mean < < "\n\n";
30:    return 0;
31: }
```

以下是清单 7.3 中程序的一个运行范例:

输　出

```
Array is:
x[0] = 12.2
x[1] = 45.4
x[2] = 67.2
x[3] = 12.2
x[4] = 34.6
x[5] = 87.4
x[6] = 83.6
x[7] = 12.3
x[8] = 14.8
x[9] = 55.5

Number of data points = 10

Mean = 42.52
```

分　析

注意清单 7.3 中的程序没有声明它的前一个版本(在清单 7.2 中)中出现的常量 MAX。程序如何决定数组元素的数目? 第 17 行示意了程序通过用数组 x 的大小(利用 sizeof(x)获得)除以第一个元素的大小(利用 sizeof(x[0])获得)计算数组 x 中元素的数目。可以用这种方法得到任何数据类型数组的大小。

7.4　函数中的数组参数

C++ 允许将函数参数声明为数组。实际上,C++ 既允许数组参数的大小为确定的,也允许为不定的。如果希望数组参数接受固定大小的数组,可以在参数描述中说明数组大小;相反,如果希望数组参数接受同样基本类型但不同大小的数组,可以在数组参数中用空方括号。

语　法

固定数组参数

固定数组参数的一般语法是:

```
type parameterName[arraySize]
```

示　例:

```
int minArray(int arr[100], int n);
void sort(unsigned dayNum[7]);
```

语　法

开数组参数

声明开数组参数的一般语法为:

```
type parameterName[]
```

示　例:

```
int minArray(int arr[], int n);
void sort(unsigned dayNum[]);
```

要在函数中使用开数组参数。

不要忘记在通用函数中检查开数组参数的上界。

我们来看一个简单的例子。清单 7.4 中列出了程序 ARRAY4.CPP 的源代码。程序完成以下任务:

- **提示用户输入数据点的数目,范围在 2 到 10 之间。**
- **提示用户输入数组的整数值。**
- **显示数组中最小值。**
- **显示数组中最大值。**

键　入　　**清单 7.4　程序 ARRAY4.CPP 的源代码**

```
1: // C++ program that passes arrays as arguments of functions
2:
3: #include <iostream.h>
4:
5: const int MAX = 10;
6:
7: main()
8: {
9:    int arr[MAX];
10:    int n;
11:
12:    // declare prototypes of functions
13:    int getMin(int a[MAX], int size);
14:    int getMax(int a[], int size);
15:
16:    do { // obtain number of data points
17:       cout << "Enter number of data points [2 to "
18:            << MAX << "] : ";
19:       cin >> n;
20:       cout << "\n";
21:    } while (n < 2 ©©¦n > MAX);
22:
23:    // prompt user for data
24:    for (int i = 0; i < n; i++) {
25:       cout << "arr[" << i << "] : ";
```

```
26:       cin > > arr[i];
27:    }
28:
29:    cout < < "Smallest value in array is"
30:          < < getMin(arr, n) < < "\n"
31:          < < "Biggest value in array is "
32:          < < getMax(arr, n) < < "\n";
33:    return 0;
34: }
35:
36:
37: int getMin(int a[MAX], int size)
38: {
39:    int small = a[0];
40:    // search for the smallest value in the
41:    // remaining array elements
42:    for (int i = 1; i < size; i+ + )
43:       if (small > a[i])
44:          small= a[i];
45:    return small;
46: }
47:
48: int getMax(int a[], int size)
49: {
50:    int big = a[0];
51:    // search for the biggest value in the
52:    // remaining array elements
53:    for (int i = 1; i < size; i+ + )
54:       if (big < a[i])
55:          big= a[i];
56:    return big;
57: }
```

以下是清单 7.4 中程序的一个运行范例:

输　出

```
Enter number of data points [2 to 10] : 5
arr[0] : 55
arr[1] : 69
arr[2] : 47
arr[3] : 85
arr[4] : 14
Smallest value in array is 14
Biggest value in array is 85
```

分　析

清单 7.4 中的程序在第 5 行声明了全程常量 MAX 以确定数据数组的大小。函数 main 在第 9 行中说明了 int 型数组 arr。第 10 行中包含了 int 型变量 n 的说明。第 13 和 14 行说明了分别返回一个 int 类型数组中最小值和最大值的函数 getMin 和 getMax 的

原型。函数 getMin 的原型表明使用了定长数组参数;相反,函数 getMax 原型表明使用了不定长数组参数。为了示范,程序中使用了两种数组参数。

第 16 行到 21 行的 do-while 循环获得希望存储在数组 arr 中的数据点的数目。第 17 和 18 行的输出语句提示用户输入数据点的数目。输出表明合法数字范围在 2 到 MAX 之间。第 19 行中的语句读取输入并存入变量 n。while 子句检查输入合法性,判断变量 n 的值是否小于 2 或大于 MAX,如果条件为真,do-while 循环再次迭代以读取正确的输入值。

第 24 到 27 行的 for 循环语句提示用户输入数据。循环使用了控制变量 i 以 1 为增量从 0 到 n- 1 进行迭代。第 25 行的输出语句提示输入所指明的数组元素的值。第 26 行的语句获得输入并存入元素 arr[i]中。

第 29 行到 32 行的输出语句显示数组 arr 中的最小和最大整数。语句以变元 arr 和 n 调用函数 getMin 和 getMax。

程序在 37 到 46 行定义函数 getMin。函数有两个参数:int 型的固定数组参数 a 以及 int 型参数 size。函数说明了局部变量 small,并用参数 a 的第一个元素 a[0]初始化。函数在第 42 行中使用 for 循环从参数 a 中查找最小值。循环中声明了控制变量 i,并以 1 为增量从 1 到 size- 1 进行迭代。循环中包含了 if 语句,如果变量 small 大于元素 a[i]的值,if 语句便把 a[i]的值赋给变量 small。函数返回变量 small 的值。getMin 函数只接受有 MAX 个元素的 int 类型的数组。

程序在第 48 到 57 行定义了函数 getMax。这个函数与函数 getMin 相似,有两个参数:int 型的开数组参数 a 和 int 型参数 size。函数说明了局部变量 big,并用参数 a 的第一个元素 a[0]初始化。函数在第 53 行中使用 for 循环从参数 a 中查找最大值。循环说明了控制变量 i,并以 1 为增量从 1 到 size- 1 进行迭代。循环中包含了 if 语句,如果变量 big 小于元素 a[i],则把元素 a[i]的值赋给变量 big。函数返回变量 big 的值。getMax 函数接受任何大小的 int 类型数组。

7.5 数 组 排 序

数组的排序和查找是数组中最常用的非数值操作。数组排序一般是按升序放置元素。这一过程要用到部分或全部元素值,这将由数组元素的先后顺序决定。显然,在有序数组中查找数据要要比在无序数组中容易得多。

计算机科学家们花费了大量的时间和精力研究提出了数组排序方法。去讨论并比较这些方法超出了本书范围。这里只提一下几种数组排序方法,包括快速排序、Shell-Metzner 排序、堆排序以及新梳形排序。一般来说,快速排序是最快的方法,但是需要一些运行开销。Shell-Metzner 排序和梳形排序方法不需要类似的开销。在这节的例子中使用了比 Shell-Metzner 排序更有效的新梳形排序方法。

对于有 N 个元素的数组 A,梳形排序方法的步骤如下:

1. 初始化用于元素比较的 Offset 的值为 N;
2. 置 Offset 的值为 8* Offset/ 11 或 1 中的较大者;
3. 设置 InOrder 标志为真;
4. 用循环控制变量 I 控制从 0 到 N- Offset 进行循环:

把 I + Offset 的赋值给 J;

如果 A[I]大于 A[J],则 A[I]与 A[J]交换并置 InOrder 标志为假;

5. 如果 Offset 不为 1 并且 InOrder 为假,重复第 2 步。

我们来看一个对整数数组排序的程序。清单 7.5 中列出了程序 ARRAY5.CPP 的源代码。程序完成以下操作:

- 提示用户输入数据点数目;
- 提示用户输入数组的整数值;
- 显示无序数组的元素;
- 显示排序数组的元素。

键　入　　清单 7.5　程序 ARRAY5.CPP 的源代码

```
1: // C++ program that sorts arrays using the Comb sort method
2:
3: #include <iostream.h>
4:
5: const int MAX = 10;
6: const int TRUE = 1;
7: const int FALSE = 0;
8:
9: int obtainNumData()
10: {
11:    int m;
12:    do { // obtain number of data points
13:       cout << "Enter number of data points [2 to "
14:            << MAX << "] : ";
15:       cin >> m;
16:       cout << "\n";
17:    } while (m < 2 ©©¦m > MAX);
18:    return m;
19: }
20:
21: void inputArray(int intArr[], int n)
22: {
23:    // prompt user for data
24:    for (int i = 0; i < n; i++) {
25:      cout << "arr[" << i << "] : ";
26:      cin >> intArr[i];
27:    }
28: }
29:
30: void showArray(int intArr[], int n)
31: {
32:    for (int i = 0; i < n; i++) {
33:       cout.width(5);
34:       cout << intArr[i] << " ";
35:    }
36:    cout << "\n";
37: }
```

```
38:
39: void showArray(int intArr[], int n)
40: {
41:    int offset, temp, inOrder;
42:
43:    offset = n;
44:    do {
45:       offset = (8 * offset) / 11;
46:       offset = (offset == 0) ? 1: offset;
47:       inOrder = TRUE;
48:       for (int i = 0; j = offset; i < (n - offset); i++, j++) {
49:          if (intArr[i] > intArr[j]) {
50:             inOrder = FALSE;
51:             temp = intArr[i];
52:             intArr[i] = intArr[j];
53:             intArr[j] = temp;
54:          }
55:       }
56:    } while (! (offset = 1 && inOrder == TRUE));
57: }
58:
59: main()
60: {
61:    int arr[MAX];
62:    int n;
63:
64:    n = obtainNumData();
65:    inputArray(arr, n);
66:    cout << "Unordered array is:\n";
67:    showArray(arr, n);
68:    sortArray(arr, n);
69:    cout << "\nSorted array is:\n";
70:    showArray(arr,n);
71:    return 0;
72: }
```

以下是清单 7.5 中程序的一个运行范例:

输　出

```
Enter number of data points [2 to 10] : 10
arr[0] : 55
arr[1] : 68
arr[2] : 74
arr[3] : 15
arr[4] : 28
arr[5] : 23
arr[6] : 69
arr[7] : 95
arr[8] : 22
arr[9] : 33
Unordered array is:
55   68   74   15   28   23   69   95   22   33
```

```
Sorted array is:
15   22   23   28   33   55   68   69   74   95
```

分　析

清单 7.5 中的程序在第 5 到 7 行声明了常量 MAX、TRUE 和 FALSE。常量 MAX 定义了程序中使用的数组大小。常量 TRUE 和 FALSE 定义了布尔值。程序还定义了函数 obtainNumData, inputArray, showArray, sortArray 和 main。

第 9 到 19 行定义的无参数函数 obtainNumData 提示用户输入值的个数。第 13 和 14 行的输出语句还表明了输入的合法范围。第 15 行的语句把输入存储到局部变量 m 中。函数使用 do-while 循环保证返回一个合法数值。只要变量 m 的值小于 2 或大于 MAX, 循环便迭代。函数返回变量 m 的值。

第 21 到 28 行定义的函数 inputArry 获得测试数组的数据。函数有两个参数。不定长数组参数 intArr 向函数调用者传回输入值。参数 n 表明参数 intArr 获得了多少值。函数使用增量为 1, 从 0 到 n- 1 迭代的 for 循环。每一次循环迭代提示输入一个值, 并存储到数组 intArr 的一个元素中。

> **注　意**　函数 inputArray 表明 C+ + 函数对待数组参数就像对待变元的引用一样, 因为这些参数可以影响在函数的作用域外的变元值。实际上 , 在处理数组参数时, C+ + 编译器传递一份数组自变量地址的拷贝给函数。掌握了数组的地址, C+ + 函数就可修改在这些函数的作用域外的数组的值。这一特性是可能的, 因为函数是在对原数组而不是其副本进行操作。

第 30 到 37 行定义的函数 showArray 显示一个数组中有意义的数据。函数有两个参数。开数组参数 intArr 向函数传递要显示的数组值; 参数 n 说明 intArr 中有多少数组元素要显示(记住, 并不是所有的数组元素都用于存储你的数据)。函数使用了以 1 为增量从 0 到 n- 1 进行迭代的 for 循环。每一次循环迭代显示一个数组元素的值。数组元素显示在同一行上。

函数 sortArray 在 39 到 57 行定义, 它使用梳形排序法对一个数组中的元素进行排序。函数有两个参数。开数组参数传递由函数进行排序的数组的值; 参数 n 表明排序的数组有多少元素。函数 sortArray 中的语句实现了早先提出的梳形排序法。

> **注　意**　函数 sortArray 表明数组参数如何进行数据传递。函数 sortArray 接收一个无序数组、排序、并把有序数组传递给函数调用者。编译器通过传递数组地址的副本支持这一性质。因此, 函数不需要明确地返回数组, 因为它是直接对原数据操作而不是数据的拷贝。

函数 main 通过调用前面提到的函数完成各种任务。函数分别在第 61 和 62 行声明数组 arr 和简单变量 n。第 62 行的语句调用 obtainNumData 获得希望在数组中存储的数据

的数目,函数 obtainNumData 的结果赋值给变量 n。第 65 行的语句调用函数 inputArray 提示输入数据。函数调用传递了自变量 arr 和 n。第 66 行的输出语句显示的信息表示程序将要显示的是未排序的数组元素。第 67 行的语句调用 showArray 并传递自变量 arr 和 n。这一函数调用在一行上显示数组元素。第 68 行的语句调用函数 sortArray 对数组 arr 中的前 n 个元素排序。第 69 行的输出语句显示的信息表明程序显示的是排序后的数组,第 70 行的语句调用 showArray 并传递自变量 arr 和 n,函数调用在同一行上显示排序后的数组 arr。

7.6　数 组 查 找

数组查找是另一个重要的非数值操作。因为数组可能是有序或无序的,因此对每一种情况的查找方法都有一种通用的分类。对于无序数组,最简单的查找方法是线性查找法;对有序数组,最简单的查找方法是二分查找法。无序数组所用的查找方法也能应用于有序数组。但是,这些方法不能利用数组有序的特点。

新　词

线性查找(linear search)

线性查找法顺序查找数组元素,寻找与给定值匹配的元素。如果查找值不在数组中,线性查找法会遍历整个数组元素。

新　词

二分查找(binary search)

二分查找法利用数组有序性的特点,通过不断缩小处理区间查找匹配值。初始查找区间包括所有数组元素(包含有意义数据的元素)。这种方法用区间的中值元素与给定值比较。如果匹配,查找停止;否则,判定哪一个子区间作为下一次查找区间。因此,每一个查找区间都是前一次的一半。如果给定值与检索数组中的元素不匹配,二分查找所做的比较比线性查找少得多。对有序的数组,二分查找法是最有效的通用查找方法。

要用无序数组的查找方法,当不能确定数组是否有序时。
不要对无序数组使用有序数组的查找方法,这种查找的结果是不可靠的。

我们来看一个对整数数组排序的例子。清单 7.6 中列出了程序 ARRAY6.CPP 的源代码。它是通过对程序 ARRAY5.CPP 添加函数和操作产生的。程序完成下列操作;

1. 提示用户输入数据点数目;
2. 提示用户输入数组的整数值;
3. 显示无序数组的元素;
4. 询问是否希望在无序数组中查找数据;
 (如果键入的字符不是 Y 或 y,程序从第 8 步继续)
5. 提示用户输入给定值;
6. 显示查找结果(如果程序找到匹配元素,显示元素的索引;否则,程序告诉用户没

有匹配值);

7. 重复第 4 步;
8. 显示有序数组的元素;
9. 询问是否希望在有序数组中查找数据(如果键入的字符不是 Y 或 y, 程序结束);
10. 提示用户输入给定值;
11. 显示查找结果(如果程序找到匹配元素, 显示元素的索引; 否则, 程序告诉用户没有匹配值);
12. 重复第 9 步。

键　入　　**清单 7.6　程序 ARRAY6.CPP 的源代码**

```
// C++ program that searches arrays using the linear
// and binary searches methods

#include <iostream.h>

const int MAX = 10;
const int TRUE = 1;
const int FALSE = 0;
const int NOT_FOUND = -1;

int obtainNumData()
{
   int m;
   do { // obtain number of data points
      cout << "Enter number of data points [2 to "
           << MAX << "] : ";
      cin >> m;
      cout << "\n";
   } while (m < 2 ©©¦m> MAX);
   return m;
}

void inputArray(int intArr[], int n)
{
   // prompt user for data
   for (int i = 0; i < n; i++) {
      cout << "arr[" << i << "] : ";
      cin >> intArr[i];
   }
}

void showArray(int intArr[], int n)
{
   for (int i = 0; i < n; i++) {
      cout.width(5);
      cout << intArr[i] << " ";
   }
   cout << "\n";
```

```
39: }
40:
41: void showArray(int intArr[], int n)
42: // sort the first n elements of array intArr
43: // using the Comb sort method
44: {
45:    int offset, temp, inOrder;
46:
47:    offset = n;
48:    do {
49:       offset = (8 * offset) / 11;
50:       offset = (offset == 0) ? 1 : offset;
51:       inOrder = TRUE;
52:       for (int i = 0; j = offset; i < (n - offset); i++, j++) {
53:          if (intArr[i] > intArr[j]) {
54:             inOrder = FALSE;
55:             temp = intArr[i];
56:             intArr[i] = intArr[j];
57:             intArr[j] = temp;
58:          }
59:       }
60:    } while (!(offset = 1 && inOrder == TRUE));
61: }
62:
63: int LinearSearch(int searchVal, int intArr[], int n)
64: // perform linear search to locate the first
65: // element in array intArr that matches the value
66: // of searchVal
67: {
68:    int notFound = TRUE;
69:    int i = 0;
70:    // search through the array elements
71:    while (i < n && not Found)
72:      // no match?
73:      if (searchVal != intArr[i])
74:         i++; // increment index to compare the next element
75:      else
76:         notFound = FALSE; // found a match
77:   // return search outcome
78:   return (notFound == FALSE) ? i : NOT_FOUND;
79: }
80:
81: int binarySearch(int searchVal, int intArr[], int n)
82: // perform binary search to locate the first
83: // element in array intArr that matches the value
84: // of searchVal
85: {
86:    int median, low, high;
87:
88:    // initialize the search range
89:    low = 0;
90:    high = n-1;
91:    // search in array
92:    do {
```

```
93:       // obtain the median index of the current search range
94:       median = (low + high) / 2;
95:       // update search range
96:       if (searchVal > intArr[median])
97:         low = median + 1;
98:       else
99:         high = median - 1;
100:    } while (! (searchVal = = intArr[median] ©©¦low > high));
101:    // return search outcome
102:    return (searchVal = = intArr[median]) ? median : NOT- FOUND;
103:  }
104:
105:  void searchInUnorderedArray(int intArr[], int n)
106:  // manage the linear search test
107:  {
108:    int x, i;
109:    char c;
110:    // perform linear search
111:    cout < < "Search in unordered array? (Y/N)";
112:    cin > > c;
113:    while (c = = 'Y' ©©¦c = = 'y') {
114:      cout < < "Enter search value : ";
115:      cin > > x;
116:      i = linearSearch(x, intArr, n);
117:      if (i ! = NOT- FOUND)
118:        cout < < "Found matching element at index " < < i < < "\n";
119:      else
120:        cout < < "No match found\n";
121:      cout < < "Search in unordered array? (Y/N) ";
122:      cin > > c;
123:    }
124:  }
125:
126:  void searchInSortedArray(int intArr[], int n)
127:  // manage the binary search test
128:  {
129:    int x, i;
130:    char c;
131:    // perform binary search
132:    cout < < "Search in sorted array? (Y/N) ";
133:    cin > > c;
134:    while (c = = 'Y' ©©¦c = = 'y') {
135:      cout < < "Enter search value : ";
136:      cin > > x;
137:      i = binarySearch(x, intArr, n);
138:      if (i ! = NOT- FOUND)
139:        cout < < "Found matching element at index " < < i < < "\n";
140:      else
141:        cout < < "No match found\n";
142:      cout < < "Search in sorted array? (Y/N) ";
143:      cin > > c;
144:    }
145:  }
146:
```

```
147: main()
148: {
149:    int arr[MAX];
150:    int n;
151:
152:    n = obtainNumData();
153:    inputArray(arr, n);
154:    cout << "Unordered array is:\n";
155:    showArray(arr, n);
156:    searchInUnorderedArray(arr, n);
157:    sortArray(arr, n);
158:    cout << "\nSorted array is:\n";
159:    showArray(arr, n);
160:    searchInSortedArray(arr, n);
161:    return 0;
162: }
```

以下是清单 7.6 中程序的一个运行范例:

输　出

```
Enter number of data points [2 to 10] : 5
arr[0]:85
arr[1]:41
arr[2]:55
arr[3]:67
arr[4]:48
Unordered array is:
  85  41  55  67  48
Search in unordered array? (Y/N) y
Enter search value : 55
Found matching element at index 2
Search in unordered array? (Y/N) y
Enter search value : 41
Found matching element at index 1
Search in unordered array? (Y/N) n
Sorted array is:
  41  48  55  67  85
Search in sorted array? (Y/N) y
Enter search value : 55
Found matching element at index 2
Search in sorted array? (Y/N) y
Enter search value : 67
Found matching element at index 3
Search in sorted array? (Y/N) n
```

分　析

清单 7.6 中的程序声明了函数 obtainNumData, inputArray, sortArray, linearSearch, binarySearch, SearchInUnorderedArray, searchInSortedArray **和** main。**因为前三个函数与清单 7.5 中的一样,所以仅讨论其余的函数。**

linearSearch 函数完成线性查找，在数组 intArr 中寻找第一个与参数 searchVal 的值相匹配的元素。函数查找数组 intArr 中的前 n 个元素。linearSearch 函数返回数组 intArr 中匹配元素的索引，如果没有找到匹配元素则返回全程常量 NOT_FOUND。函数用 while 循环测试数组 intArr 的元素。当变量 i 的值小于变量 n 并且局部变量 notFound 为 TRUE 时，查找循环便迭代。第 78 行的语句利用条件运算符返回函数结果。

binarySearch 函数与 linearSearch 函数的参数相同并返回同样的值。函数用局部变量 low 和 high 存储当前查找区间。函数分别用 0 和 n- 1 初始化变量 low 和 high。第 92 到 100 行的 do-while 循环计算中值元素的索引并用中值元素与查找值比较。第 96 行的 if 语句进行这一比较，并且它的子句依据比较结果修改变量 low 或变量 high 的值。对变量的修改缩小了查找区间。第 102 行的返回语句依据查找值与当前查找区间的中值元素的比较结果产生函数值。

函数 searchInUnorderedArray 管理对无序数组的查找。函数使用开数组参数 intArr 访问无序数组。函数说明了用于存储查找值的局部变量。第 116 行的语句调用函数 linearSearch 并传递参数 x(存储查找值的局部变量)，intArr 和 n，并把函数 linearSearch 的结果赋给局部变量 i。第 117 行的 if 语句变量 i 的值是否非 NOT_FOUND。如果这个条件为真，第 118 行的输出语句显示匹配元素的索引；否则，第 120 行的输出语句显示没有找到匹配元素的信息。

函数 searchInSortedArray 与函数 searchInUnorderedArray 很相似。主要的不同之处是函数 searchInSortedArray 处理有序数组，因此调用 binarySearch 函数在有序数组 intArr 中进行二分查找。

函数 main 调用这些函数，以支持前面描述的程序任务。

7.7　多 维 数 组

在多维数组中，每增加一个维数都增加一个访问索引。二维数组(或矩阵)是最常用的一种多维数组。三维数组则比矩阵用得少，余此类推。

新　词

多维数组(multidimensional array)
多维数组是一维数组的超集。

语　法

二维和三维数组
声明二维和三维数组的一般语法为：

```
type array [size1][size2];
type array [size1][size2][size3];
```

对于简单数组，每一维都以 0 为下界下标，并且声明定义了每一维元素的个数。

示　例：

```
double matrixA[100][10];
char table [41][22][3];
int index [7][12];
```

了解一下 C++ 如何存储多维数组元素是很重要的。大多数编译器以邻接方式存储多维数组元素(作为一个长数组)。运行时代码计算要定位的元素在长数组中的位置。为

说明多维数组的存储表,让我们从不同维的索引引用的转换开始。下表说明了维数和高维、低维的概念。下面是一个六维数组——一种极端情况的例子:

```
  1       2       3       4       5       6   <- - - - 维数
M[20]    [7]     [5]     [3]     [2]     [2]
 高维   - - - >
```

数组 M 的第一个元素是 M[0][0][0][0][0][0],并存储在数组 M 的第一个存储位置。数组 M 存储在有 8400 个元素的连续块中。连续块中的下一个位置存储存储最高维即第 6 维索引为 1 的元素(M[0][0][0][0][0][1]),而 1 已经是第 6 维的上限,到达这一界限使第 5 维的下标变为 1 并且第 6 维的下标重置为 0。这一过程重复到多维数组中的每一个元素都被访问到。这一存储方案可以描述为汽车加油时的加油泵计量表:最右边的数字转得最快,左边的数字转得最慢。

下面是使用三维数组的一个例子,M[3][2][2]:

```
M[0][0][0]   ←起始存储位置
M[0][0][1]   ←第 3 维填满
M[0][1][0]
M[0][1][1]   ←第 2 维和第 3 维填满
M[1][0][0]
M[1][0][1]   ←第 3 维填满
M[1][1][0]
M[1][1][1]   ←第 2 维和第 3 维填满
M[2][0][0]
M[2][0][1]   ←第 3 维填满
M[2][1][0]
M[2][1][1]   ←所有维数均已填满
```

我们来看一个说明基本矩阵运算的例子。清单 7.7 中列出了 MAT1.CPP 的源代码。程序管理具有 10 列 30 行的矩阵并完成下列任务:

- 提示用户输入行数(程序验证输入合法性);
- 提示用户输入列数(程序验证输入合法性);
- 提示输入矩阵元素;
- 计算显示矩阵每一列的平均值。

键　入　　清单 7.7　程序 MAT1.CPP 的源代码

```
1: /*
2:   C++ program that demonstrates the use of two- dimension arrays.
3:   The average value of each matrix column is calculated.
4: */
5:
6: #include <iostream.h>
```

```
 7:
 8: const int MAX_COL = 10;
 9: const int MAX_ROW = 30;
10:
11: main()
12: {
13:     double x[MAX_ROW][MAX_COL];
14:     double sum, sumx, mean;
15:     int rows, columns;
16:
17:     // get the number of rows
18:     do {
19:       cout << "Enter number of rows [2 to "
20:            << MAX_ROW << "] : ";
21:       cin >> rows;
22:     } while (rows < 2 || rows > MAX_ROW);
23:
24:     // get the number of columns
25:     do {
26:       cout << "Enter number of columns [1 to "
27:            << MAX_COL << "]:";
28:       cin >> columns;
29:     } while (columns < 1 || columns > MAX_COL);
30:
31:     // get the matrix elements
32:     for (int i = 0; i < rows; i++)  {
33:       for (int j = 0; j < columns; j++)  {
34:           cout << "X[" << i << "][" << j << "] : ";
35:           cin >> x[i][j];
36:       }
37:       cout << "\n";
38:     }
39:
40:     sum = rows;
41:     // obtain the sum of each column
42:     for (int j = 0; j < columns; j++)  {
43:       // initialize summations
44:       sumx = 0.0;
45:       for (i = 0; i < rows; i++)
46:         sumx += x[i][j];
47:       mean = sumx / sum;
48:       cout << "Mean for column " << j
49:            << " = " << mean << "\n";
50:       }
51:       return 0;
52: }
```

以下是清单 7.7 中程序的一个运行范例:

输　出

```
Enter number of rows [2 to 30] : 3
Enter number of columns [1 to 10] : 3
x[0][0] : 1
x[0][1] : 2
x[0][2] : 3

x[1][0] : 4
x[1][1] : 5
x[1][2] : 6

x[2][0] : 7
x[2][1] : 8
x[2][2] : 9

Mean for column 0 =  4
Mean for column 1 =  5
Mean for column 2 =  6
```

分　析

清单 7.7 中的程序分别在第 8 和第 9 行声明了全局常量 MAX_COL 和 MAX_ROW,它们定义了程序中所创建的矩阵的维数。函数 main 声明了有 MAX_ROW 行和 MAX_COL 列的矩阵 x,还声明了其他非数组变量。

第 18 到 22 行的 do-while 循环提示用户输入存储数据的矩阵 x 的行数。第 19 和 20 行的输出语句表明行数的有效范围。第 21 行语句把输入存放在变量 rows 中。

第 25 到 29 行的第二个 do-while 循环提示用户输入存储数据的矩阵 x 的列数。第 26 和 27 行的输出语句指明列数的有效范围,第 28 行语句把输入存放在变量 columns 中。

第 32 到 38 行的嵌套 for 循环提示用户输入矩阵元素。外层 for 循环使用控制变量 i 并以 1 为增量从 0 到 rows- 1 进行迭代;内层 for 循环使用控制变量 j 以 1 为增量从 0 到 columns- 1 进行迭代。第 34 行的输出语句显示将要接收输入的矩阵元素的索引。第 35 行的语句把输入存储到矩阵元素 x[i][j]中。

从第 40 行开始求矩阵每一列的平均值。该行的语句把变量 rows 中的整数赋值给 double 型的变量 sum。程序在第 42 到 50 行使用了另一个嵌套的 for 循环。外层 for 循环使用控制变量 j 以 1 为增量从 0 到 columns- 1 进行迭代,此循环对每一列进行处理。外层循环中的第一条语句是对变量 sumx 赋值 0。位于第 45 行的内层 for 循环使用了控制变量 i 并以 1 为增量从 0 到 rows- 1 进行迭代。内层循环利用第 46 行的语句将元素 x[i][j]的值累加到变量 sumx 中。第 47 行的语句(在内层 for 循环之外)计算列的平均值并赋值给 mean。第 48 和 49 行的输出语句显示列号和它的平均值。

注　意　第 42 行的 for 循环重新声明了控制变量 j(第 45 行的 for 循环没有这样做)。为什么? 第 33 行的 for 循环虽然也声明了控制变量 j, 但是该循环的作用域被限制于外层 for 循环内, 一旦第一对嵌套循环执行完成, 循环控制变量 j 便被运行系统删除。

7.8　初始化多维数组

C++ 允许用与一维数组相似的方式初始化多维数组。这需要使用一个与被初始化的多维数组元素的存储有相同顺序的取值表。现在你可以意识到了解 C++ 如何存储多维数组元素的重要性了。对前面的 C++ 程序做一下修改, 使用内部给程序提供数据的初始化表。因此, 程序不需要输入任何数据, 但将显示矩阵值和各列的平均值。

键　入　　**清单 7.8　程序 MAT2.CPP 的源代码**

```
 1: /*
 2:   C++ program that demonstrates the use of two- dimension arrays.
 3:   The average value of each matrix column is calculated.
 4: */
 5:
 6: #include <iostream.h>
 7:
 8: const int MAX_COL =  3;
 9: const int MAX_ROW =  3;
10:
11: main()
12: {
13:      double x[MAX_ROW][MAX_COL] =  {
14:                                        1, 2, 3, // row #  1
15:                                        4, 5, 6, // row #  2
16:                                        7, 8, 9  // row #  3
17:                                        };
18:      double sum, sumx, mean;
19:      int rows =  MAX_ROW, columns =  MAX_COL;
20:
21:      cout << "Matrix is:\n";
22:      // display the matrix elements
23:      for (int i =  0; i <  rows; i++ )  {
24:        for (int j =  0; j <  columns; j++ )  {
25:             cout.width(4);
26:             cout.precision(1);
27:             cout << x[i][j] << " ";
28:        }
29:      cout << "\n";
30:   }
31:   cout << "\n";
32:
```

```
33:    sum = rows;
34:    // obtain the sum of each column
35:    for (int j = 0; j < columns; j++)  {
36:       // initialize summations
37:       sumx = 0.0;
38:       for (i = 0; i < rows; i++)
39:          sumx += x[i][j];
40:       mean = sumx / sum;
41:       cout << "Mean for column" << j
42:            << " = " << mean << "\n";
43:    }
44:    return 0;
45: }
```

以下是清单 7.8 中程序的一个运行范例:

输　出

```
Matrix is:
1    2    3
4    5    6
7    8    9

Mean for column 0 = 4
Mean for column 1 = 5
Mean for column 2 = 6
```

分　析

清单 7.8 中的程序声明了矩阵 x 并用取值表进行初始化。注意,程序声明的常量 MAX_COL 和 MAX_ROW 的值与被初始化的矩阵的大小一致。第 13 到 17 行的声明语句显示每一行赋值的元素。函数 main 还用常量 MAX_ROW 和 MAX_COL 分别初始化变量 rows 和 columns。执行这一初始化有两个原因:首先,程序不再要求输入变量 row 和 columns 的值;其次,程序对适应专门大小的矩阵 x 进行操作。

第在 21 到 30 行的程序使用了嵌套的 for 循环显示矩阵 x 的元素。第二对嵌套的 for 循环计算矩阵每一列的平均值。这一嵌套 for 循环与清单 7.7 中的相同。

7.9　多维数组参数

C++ 允许把函数参数声明为多维数组。像一维数组那样,C++ 允许数组参数的大小可以是明确的或不定的。但是,对后一种情况,则只能是数组的第一维为不确定的。如果希望数组参数接受固定维数的数组,可以在参数描述中说明数组每一维的大小;相反,如果希望数组参数接受具有相同基本类型但第一维大小不同的数组,则在数组参数的第一维用空方括号说明。

语　法

固定数组参数

固定数组的一般语法是:

```
type parameterName[dim1Size][dim2Size]...
```

示　例:

```
int minMatrix(int intMat[100][20], int rows, int cols);
void sort (unsigned mat[23][55],
          int rows, int cols, int colindex);
```

语　法

开数组参数

声明开数组参数的一般语法是:

```
type parameterName[][dim2Size]...
```

示　例:

```
int minMatrix(int intMat[][100], int rows, int cols);
void sort(unsigned mat[][55],
          int rows, int cols, int colIndex);
```

我们来看一个例子。清单 7.9 中列出了程序 MAT3.CPP 的源代码。程序执行与清单 7.7 中程序 MAT1.CPP 相同的任务。通过修改程序 MAT1.CPP, 并把每一个程序任务放在一个独立的函数中创建了程序 MAT3.CPP。这样, 程序 MAT3.CPP 是程序 MAT1.CPP 的高度结构化的版本。

键　入　　**清单 7.9　程序 MAT3.CPP 的源代码**

```
 1: /*
 2:   C++ program that demonstrates the use of two- dimension arrays.
 3:   The average value of each matrix column is calculated.
 4: */
 5:
 6: #include <iostream.h>
 7:
 8: const int MAX_COL =  10;
 9: const int MAX_ROW =  30;
10:
11: int getRows()
12: {
13:    int n;
14:    // get the number of rows
15:    do {
16:       cout << "Enter number of rows [2 to "
17:            << MAX_ROW << "] : ";
18:       cin >> n;
19:    } while (n < 2 ©Ð¦n > MAX_ROW);
20:    return n;
21: }
22:
23: int getColumns()
24: {
```

```
25:    int n;
26:    // get the number of columns
27:    do {
28:       cout < < "Enter number of columns [1 to "
29:            < < MAX - COL < < "] : ";
30:       cin > > n;
31:    } while (n < 1 ©©¦n > MAX - COL);
32:    return n;
33: }
34:
35: void inputMatrix(double mat[][MAX - COL],
36:                    int rows, int columns)
37: {
38:    // get the matrix elements
39:    for (int i = 0; i < rows; i+ + )  {
40:       for (int j = 0; j < columns; j+ + )  {
41:          cout < < "X[" < < i < < "][" < < j < < "]:";
42:          cin > > mat[i][j];
43:       }
44:       cout < < "\n";
45:    }
46: }
47:
48: void showColumnAverage(double mat[][MAX - COL],
49:                        int rows, int columns)
50: {
51:    double sum, sumx, mean;
52:    sum = rows;
53:    // obtain the sum of each column
54:    for (int j = 0; j < columns; j+ + ) {
55:       // initialize summations
56:       sumx = 0.0;
57:       for (i = 0; i < rows; i+ + )
58:          sumx + = mat[i][j];
59:       mean = sumx / sum;
60:       cout < < "Mean for column" < < j
61:            < < " = " < < mean < < "\n";
62:    }
63: }
64:
65: main()
66: {
67:       double x[MAX - ROW][MAX - COL];
68:       int rows, columns;
69:       // get matrix dimensions
70:       rows = getRows();
71:       columns = getColumns();
72:       // get matrix data
73:       inputMatrix(x, rows, columns);
74:       // show results
75:       showColumnAverage(x, rows, columns);
76:       return 0;
77: }
```

以下是清单 7.9 中程序的一个运行范例:

输　出

```
Enter number of rows [2 to 30] : 3
Enter number of columns [1 to 10] : 3
X[0][0]:10
X[0][1]:20
X[0][2]:30

X[1][0]:40
X[1][1]:50
X[1][2]:60

X[2][0]:70
X[2][1]:80
X[2][2]:90

Mean for column 0 =  40
Mean for column 1 =  50
Mean for column 2 =  60
```

分　析

清单 7.9 中的程序声明了函数 getRows, getColumns, inputMatrix, showColumnAverage 和 main。函数 getRows 提示用户输入所用使用的矩阵的行数。函数返回通过合法性检查的输入值。与此相似,函数 getColumns 返回通过合法性检查的矩阵列数。

函数 inputMatrix 获得矩阵的数据。函数带三个参数。参数 mat 指定矩阵的参数(第一维空着); 参数 rows 和 columns 分别指定将要接收输入数据的矩阵 mat 的行数和列数。

该函数包含与程序 MAT1.CPP 中相同的语句。程序 MAT3.CPP 利用这些函数作为完成各种任务的语句的外壳或包容。从结构化程序设计的观点来看,程序 MAT3.CPP 优于程序 MAT1.CPP。

函数 main 说明了有 MAX_ROW 行和 MAX_COL 列的矩阵 x,并调用函数 getRows 和 getColumns 获得工作矩阵的行数和列数。第 73 行语句调用函数 inputMatrix 并提供了参数 x, rows 和 columns。第 75 行的语句调用函数 showColumnAverage 并传递了参数 x, rows, 和 columns。

小　　结

今天的课程讲述了数组操作的各种主题,包括一维数组和多维数组。主要内容有:

- 声明一维数组要说明数组元素的数据类型、数组的名字和数组元素的数目(用大括号括起来)。所有的 C++ 数组均以 0 为下界,数组的上界等于元素个数减 1。
- 使用一维数组要说明数组名字并包含有有效的索引,封闭用的大括号。

· 声明一维数组时可以做初始化。初始化数据表封闭在大括号中并且数据用逗号分隔。C+ + 允许包含少于数组元素个数的数据,在这种情况下,编译器对没有明确初始化的元素赋值 0。另外,C+ + 允许初始化的数组没有明确的大小并用初始化项数作为数组元素项数。
· 声明一维数组作为函数参数可采用两种方式。第一种方式处理固定数组参数;第二种方式处理开数组参数。固定数组参数在参数中包含了数组大小。这种参数的变元必须要与参数的类型和大小相匹配。开数组参数使用空方括号表明该参数的变元可以是任意大小。
· 数组排序是数组的重要的非数值操作。数组元素的排序顺序或者为升序或者为降序。排好序的数组查找起来更容易。对于数组排序,新梳形排序是很有效的。
· 数组查找是找到包含与查找值相同数据的数组元素的位置。查找方法取决于数组是有序的还是无序的。线性查找法适用于无序数组,二分查找法适用于有序数组。
· 声明多维数组要求说明数组元素的数据类型、数组的名字和每一维的大小(用方括号括起)。每一维的最小索引为 0,每一维的上界等于这一维的大小减 1。
· 使用多维数组要求说明数组的名字并具有合法的索引。每一个索引必须用方括号单独封闭隔开。
· 在声明多维数组时可以进行初始化。初始化数据表括在大括号中并且数据用逗号分隔。C+ + 允许数据个数少于数组大小,在这种情况下,编译器对没有明确初始化的元素赋值 0。
· 声明多维数组作参数可采用两种方式。第一种方式处理固定数组参数;第二种方式处理开放第一维参数。固定数组参数在数组参数中包含了每一维的大小。这种参数的变元必须与参数的类型和大小相匹配。开数组参数仅在第一维用空方括号表明参数对应的第一维的大小可变。变元的其他维必须与数组参数匹配。

问 与 答

问:C+ + 中允许改变数组的大小吗?

答:不允许。C+ + 不允许重定义数组的大小。

问:为创建缓冲区,可以把数组声明为基本类型 void 吗?(例如,void array[81];)

答:不能。C+ + 不允许用 void 类型定义数组。因为类型 void 没有定义大小。用 char 或 unsigned char 类型创建用做缓冲区的数组。

问:C+ + 允许重复声明一个数组吗?

答:如果在嵌套语句块中声明了数组,为改变它们的基本类型、维数和大小,C+ + 允许再次声明一个数组。下面是一个例子:

```
# include < iostream. h>
const MAX =  100;
const MAX_ROWS =  100;
const MAX_COLS =  20;
main()
```

```
{
   // declare variables here?
   {
      double x[MAX];
      // declare other variables?
      // statements to mainpulate the single-dimensional array x
   }
  {
  double x[MAX_ROWS][MAX_COLS];
  // declare other variables?
  // statements to manipulate the matrix x
  }
  return 0;
  }
```

函数 main 在第一个嵌套语句块中声明了数组 x。当程序执行到块结尾时,运行系统删除了数组 x 和块中说明的其他变量。然后在第二块中函数再次声明 x 为一个矩阵。当程序执行到第二个块结尾,运行系统删除了矩阵 x 和块中说明的其他变量。

问:数组仅限于预定义类型吗?

答:不。C++ 允许用自定义类型建立数组(参见第 8 天)。

作　　业

作业部分所提供的测验题有助于巩固理解所学内容,通过练习可以积累运用所学知识的经验。要尽量弄懂测验和练习的答案再继续下一天的课程。附录 B 的“答案”中提供了解答。

测　验

1. **以下程序的输出是什么?**

```
# include <iostream.h>
const int MAX = 5;
main()
{
  double x[MAX];
  x[0] = 1;
  for (int i = 1; i < MAX; i++)
  x[i] = i * x[i-1];
  for (i = 0; i < MAX; i++)
    count << "x[" << i << "]= " << x[i] << "\n";
 return 0;
}
```

2. **以下程序的输出是什么?**

```
# include <iostream.h>
# include <math.h>
const int MAX = 5;
main()
{
```

```
  double x[MAX];
  for (int i = 0; i < MAX; i++)
    x[i] = sqrt(double(i));
  for (i = 0; i < MAX; i++)
    count << "x[" << i << "]= " << x[i] << "\n";
  return 0;
}
```

3. **以下程序错在哪里?**

```
#include <iostream.h>
const int MAX = 5;
main()
{
  double x[MAX];
  x[0] = 1;
  for (int i = 0; i < MAX; i++)
    x[i] = i * x[i-1];
  for (i = 0; i < MAX; i++)
    count << "x[" << i << "]= " << x[i] << "\n";
  return 0;
}
```

练　习

通过修改程序 ARRAY6.CPP **编写程序** ARRAY7.CPP。**用** Shell-Metzner **方法代替函数** sortArray **中的新梳形排序方法**。

第1周

内容复习

在继续第 2 周的 Visual C+ + 程序设计学习之前，我们来看一个特殊的例子，它将在下周周末再作改进。例子是一个简单的猜数游戏，如清单 R1.1 所示。程序选择一个 0 到 1000 之间的随机数，并提示你输入一个在此范围内的数。如果输入大于未知数，程序告诉你猜得太大了；相反，如果输入小于未知数，程序告诉你猜得太小了。如果猜中了，游戏以你的胜利而告终。程序允许你猜 11 次，并可以在任何提示处输入一个负整数结束游戏。在这种情况下，游戏停止并显示未知数。

键　入　　**清单 R1.1　程序 GAME1.CPP 的源代码**

```
#include <stdlib.h>
#include <iostream.h>
#include <time.h>

// declare a global random number generating function
int random(int maxVal)
{ return rand() % maxVal; }


main()
{
   int n, m;
   int MaxIter = 11;
   int iter = 0;
   int ok = 1;


   // reseed random- number generator
   srand((unsigned)time(NULL));
   n = random(1001);
   m = -1;

   // loop to obtain the other guesses
   while (m != n && iter < MaxIter && ok == 1) {
      cout << "Enter a number between 0 and 1000 : ";
      cin >> m;
      ok = (m < 0) ? 0 : 1;
      iter++;
      // Is the user's guess higher?
      if (m > n)
         cout << "Enter a lower guess\n\n";
```

```
32:       else if (m < n)
33:          cout << "Enter a higher guess\n\n";
34:       else
35:          cout << "You guessed it! Congratulations.";
36:    }
37:    // did the user guess the secret number
38:    if (iter >= MaxIter ©©ok == 0)
39:       cout << "The secret number is " << n << "\n";
40:
41:    return 0;
42: }
```

以下是清单 R1.1 中程序的一个运行范例:

输　出

```
Enter a number between 0 and 1000 : 500
Enter a lower guess

Enter a number between 0 and 1000 : 250
Enter a higher guess

Enter a number between 0 and 1000 : - 1
Enter a higher guess

The secret number is 399
```

分　析

清单 R1.1 中的程序声明了函数 random, 以返回一个 0 到 1000 之间的随机数。程序还声明了产生猜数游戏的函数 main。函数在第 12 到 15 行声明了一些局部变量。第 19 行的语句设置随机数发生器。第 20 行的语句把要猜的数赋值给变量 n。第 21 行的语句把存储猜测值的变量 m 赋值为- 1。

第 24 到 36 行的 while 循环产生游戏。while 循环判断下列条件是否全为真:

- **猜测值(存储在 m 中)与存储在变量 n 中的未知数不一致;**
- **迭代次数(存储在变量 iter 中)小于最大迭代次数(存储在变量 MaxIter 中);**
- **变量 ok 的值为 1。**

循环中的第一条语句提示输入一个 0 到 1000 之间的数。第 26 行的语句获得输入并存到变量 m 中。如果输入了一个负整数, 第 27 行的语句把 ok 赋值为 0; 否则, 赋值为 1。第 28 行的语句增加变量 iter 的值。

第 30 到 35 行中有多个选择的 if 语句把输入与未知数进行比较并显示反映猜测情况的相应信息。

如果在 MaxIter 次重复中猜测失败或输入了一个负整数, 第 38 行的 if 语句显示出未知数。

第2周

概要

第 2 周继续学习 C+ + 语言, 内容涉及到 C+ + 的高级部分。你将学习用户自定义数据类型(尤其是结构) 和指针。这周内还将更深入地讲述函数并介绍 C+ + 中面向对象的编程方法(OOP), 学习有关类、成分以及使用这些成分的规则。另外还将学习用 C+ + 流库进行基本文件 I/O。本周最后两天将介绍 Windows 编程和 Microsoft 基本类库(MFC)。这周末, 你将编写一个简单、实用的 Windows 程序。

第8天

用户定义类型和指针

创建用户定义的数据类型是人们期望当今程序设计语言所应具有的特性之一。今天的课程学习能够更好地组织数据的枚举数据类型和结构,另外还将讨论使用简单变量、数组、结构和动态数据的指针。今天学习的主题如下:

- 使用 typedef 的类型定义
- 枚举数据类型
- 结构
- 联合
- 引用变量
- 指向现有变量的指针
- 数组指针
- 结构指针
- 用指针管理和存取动态数据
- 远指针

8.1 C++ 中的类型定义

C++ 提供了 typedef 关键字定义新的数据类型名以作为已有类型的别名。

语 法 typedef 关键字

使用 typedef 的一般语法为:

```
typedef knownType newType;
```

示 例:

```
typedef unsigned word;
typedef unsigned char byte;
typedef unsigned char boolean;
```

typedef 关键字从一个已知的类型定义新类型。用 typedef 可以创建已有数据类型名的简写或定义自己更熟悉的数据类型名(第二个例子定义了 byte 类型)。另外,typedef 语句可以定义更恰当地描述所用数据的数据类型名。以上三个例子说明了 typedef 的使用。还可以使用 typedef 定义数组类型的名字。

语　法

数组类型名

定义数组类型名的一般语法为:

```
typedef baseType arrayTypeName [array Size];
```

typedef 语句定义了 arrayTypeName, 其基本类型和大小分别为 baseType 和 arrayType。

示　例:

```
typedef double vector [10];
typedef double matrix[10][30];
```

这样, 标识符 vector 和 matrix 就成为数据类型名。

8.2　枚举数据类型

枚举数据类型遵循的规则是枚举标识符必须唯一, 而对它们的赋值不唯一。

新　词

枚举类型(enumerated type)

枚举类型定义了唯一标识符的列表和与这些标识符相关的值。

语　法

枚举类型

声明枚举类型的一般语法为:

```
enum enumType {< list of enumerated identifiers> };
```

示　例:

```
enum Boolean { false, true };
enum YesNo {no, yes, dontCare, maybe};
enum weekday { Sunday, Monday, Tuesday,
               Wednesday, Thursday, Friday, Saturday };
```

下面是声明枚举类型的例子:

```
enum CPUtype {i8088, i80286, i80386DX, i80386SX, i80486DX, i80486SX };
```

C+ + 将枚举标识符与整数相对应。例如, 在以上类型中, 编译器对 i8088 赋值 0, i80286 赋值 1, 等等。

在声明枚举类型时 C+ + 很灵活。首先, 允许对声明的枚举标识符进行显式地赋值。下面是一个例子:

```
enum weekday { Sunday = 1, Monday, Tuesday, Wednesday,
               Thursday, Friday, Saturday };
```

此声明中将枚举标识符 Sunday 显式赋值为 1, 编译器将下一个标识符赋值为 2……以此类推。C+ + 允许对枚举表中的每一个成员进行显式赋值, 更进一步, 这些值不必是唯一的。下面是灵活声明 C+ + 枚举类型的一些例子:

```
// explicit value assignment for every list member
enum colors { black = 1, red = 2, blue = 3, green = 5,
              yellow = 7, white = 11 };
```

```
// intermittent value assignment
enum colors { black = 1, red, blue, green = 5,
              yellow = 7, white = 11 };

// duplicate values
enum CPUtype { i8088 = 1, i80286 = 2,
               i80386DX = 3, i80386SX = 3,
               i80486DX = 4, i80486SX = 4};

enum choiceType { false, true, dontCare = 0 };
```

在最后一个例子中,缺省情况下编译器将标识符 false 与 0 相对应。但是,编译器还将 dontcare 与 0 相对应,因为它是显式赋值。

C++ 中允许用下述方法声明枚举类型的变量:

1. 枚举类型的说明可以包含变量类型的说明,一般语法为:

```
enum enumType {< list of enumerated identifiers> }
< list of variables> ;
```

以下是一个例子:

```
enum weekDay {Sum = 1, Mon, Tue, Wed, Thu, Fri, Sat}
              recycleDay, payDay, movieDay;
```

2. 若单独声明枚举类型及其变量,则要有多条声明类型和相关变量的语句:

```
dnum enumType {< list of enumerated identifiers> };
enumType var1, var2, …, varN;
```

我们来看一个例子,清单 8.1 中列出了程序 ENUM1.CPP 的源代码。程序实现了一个简单的单行四功能计算器完成以下任务:

- 提示用户输入一个数、一个运算符(+,-,*,或/)以及另一个数;
- 如果有效,执行所要求的运算;
- 如果运算有效则显示操作数、运算符和结果;否则显示表明错误类型的错误信息(或者是输入了错误算符,或者是除数为 0)。

键　入　　清单 8.1　程序 ENUM1.CPP 的源代码

```
1: /*
2:    C++ program that demonstrates enumerated types
3: */
4:
5: #include <iostream.h>
6:
7: enum mathError { noError, badOperator, divideByZero };
8:
9: void sayError(mathError err)
10: {
11:    switch (err) {
12:       case noError:
```

```
13:         cout << "No error";
14:         break;
15:       case badOperator:
16          cout << "Error: invalid operator";
17:         break;
18:       case divideByzero:
19:         cout << "Error: attempt to divide by zero";
20:     }
21: }
22:
23: main()
24: {
25:     double x, y, z;
26:     char op;
27:     mathError error = noError;
28:
29:     cout << "Enter a number, an operator, and a number : ";
30:     cin >> x >> op >> y;
31:
32:     switch (op) {
33:       case '+':
34          z = x + y;
35:         break;
36:       case '-':
37:         z = x - y;
38:         break;
39:       case '*':
40:         z = x * y;
41:         break;
42:       case '/':
43:         if (y != 0)
44:           z = x / y;
45:         else
46:           error = divideByZero;
47:         break;
48:       default:
49:         error = badOperator;
50:     }
51:
52:     if (error == noError)
53:       cout << x << " " << op << " " << y << " = " << z;
54:     else
55:       sayError(error);
56:     return 0;
57: }
```

以下是清单 8.1 中程序的一个运行范例:

输　出

```
Enter a number, an operator, and a number: 355 / 113
355 / 113 = 3.14159
```

分　析

清单 8.1 中程序在第 7 行声明了枚举类型 mathError。此数据类型有三个枚举值: noError, badOperator 和 divide ByZero。

程序还在第 9 到 21 行定义了函数 sayError 显示基于枚举型参数 err 值的信息, 函数使用 switch 语句显示对应各枚举值的相应信息。

函数 main 中声明了 double 型的变量 x, y 和 z 分别代表运算数和结果。另外, 函数声明了 char 型的变量 op 存储要求的运算, 还有枚举型变量 error 存储错误状态。函数用枚举型值 noError 初始化变量 error。

第 29 行的输出语句提示用户输入运算数和运算符。第 30 行的语句把输入依次存入变量 x, op 和 y。函数在第 32 行使用 switch 语句测试变量 op 的值并执行要求的操作。第 33、36、39 和 42 行的 case 标号提供了所支持的四种数学运算。最后的 case 标号中包含的 if 子句判定除数是否为 0。如果条件为真, else 子句中的语句把变量 error 赋值为枚举值 divideByZero。

第 48 行的 default 子句处理非法运算符。第 49 行语句将变量 error 赋值为枚举值 badOperator。

第 52 行的 if 语句判定变量 error 是否具有枚举值 noError。如果条件为真, 程序执行第 53 行的输出语句, 显示运算数、运算符和结果; 否则程序执行 else 子句中的语句, 以 error 为变元调用函数 sayError, 函数调用显示标识错误的信息。

8.3　结　　构

C++ 支持结构, 结构成员可以是预定义的类型或结构。

新　词

结构(structure)

结构定义了从逻辑上将几个域或成员组织起来的新类型。

语　法

结构

声明结构的一般语法为:

```
struct structTag{
<list of members>
};
```

示　例:

```
struct point {
double x;
double y;
};

struct rectangle {
    point upperLeftCorner;
    point lowerRightCorner;
    double area;
```

```
};

struct circle {
point center;
double radius;
double area;
};
```

一旦定义了一个 struct 类型,便可以使用这个类型声明变量。下面便是用上述定义的结构声明变量的例子:

```
point p1, p2, p3;
```

在定义结构时也可以声明结构变量;

```
struct point {
    double x;
    double y;
    } p1, p2, p3;
```

新　词

无标记结构(untagged structure)
无标记结构允许不定义结构的名字而声明结构变量。

注　意　有趣的是,C+ + 允许说明无标记结构。例如下面的结构定义声明了变量 p1, p2 和 p3,但略去了结构的名字:

```
struct {
      double x;
      double y;
      } p1, p2, p3;
```

C+ + 允许声明并初始化一个结构变量。以下是一个例子:

```
point pt = {1.0, -8.3};
```

使用点运算符可存取结构成员,以下是几个例子:

```
p1.x = 12.45;
p1.y = 34.56;
p2.x = 23.4 / p1.x;
p2.y = 0.98 * p1.y;
```

让我们看一个例子。清单 8.2 中列出了程序 STRUCT1.CPP 的源代码。程序提示用户用 4 组坐标点定义 4 个矩形。每一个矩形都由左上角和右下角的 X 和 Y 坐标定义。程序计算每个矩形的面积并依照面积排序,然后依次显示矩形。

键　入　　**清单 8.2　程序 STRUCT1.CPP 的源代码**

```
1: /*
2:    C++ program that demonstrates structured types
3: */
4:
```

```
 5: # include < iostream. h>
 6: # include < stdio. h>
 7: # include < math. h>
 8:
 9: const MAX - RECT =  4;
10:
11: struct point {
12:    double x;
13:    double y;
14: };
15:
16: struct rect {
17:    point ulc; // upper left corner
18:    point lrc; // lower right corner
19:    double area;
20:    int id;
21: };
22:
23: typedef rect rectArr[ MAX - RECT];
24:
25: main( )
26: {
27:    rectArr r;
28:    rect temp;
29:    double length, width;
30:
31:    for (int i =  0; i <  MAX - RECT; i+ + } {
32:       cout < <  "Enter (X, Y) coord. for ULC of rect. # "
33:            < <  i < <  " : ";
34:       cin > >  r[i]. ulc. x > >  r[ i]. ulc. y;
35:       cout < <  "Enter (X, Y) coord. for LRC of rect . #  '
36:            < <  i < <  " : ";
37:       cin > >  r[i] . lrc. x > >  r[i]. lrc. y;
38:       r[i]. id =  i;
39:       length =  fabs(r[i]. ulc. x -  r[i]. lrc. x);
40:       width =  fabs(r[i]. ulc. y -  r[i]. lrc. y);
41:       r[i]. area =  length *  width;
42:    }
43:
44:    // sort the rectangles by areas
45:    for (i =  0; i <  (MAX - RECT -  1); i+ + )
46:       for (int j =  i +  1; j <  MAX - RECT; j+ + )
47:          if (r[ i]. area >  r[j]. area ) {
48:             temp =  r[i];
49:             r[i] =  r[j];
50:             r[j] =  temp;
51:          }
52:
53:    // display rectangles sorted by area
54:    for (i =  0; i <  MAX - RECT; i+ + )
55:       printf("Rect #  %d has area %5. 14f\n", r[i]. id, r[i]. area);
56:    return 0;
57: }
```

以下是清单 8.2 中程序的一个运行范例:

输　出

```
Enter (X,Y) coord. for ULC of rect. #  0: 1 1
Enter (X,Y) coord. for LRC of rect. #  0: 2 2
Enter (X,Y) coord. for ULC of rect. #  1: 1.5 1.5
Enter (X,Y) coord. for LRC of rect. #  1: 3 4
Enter (X,Y) coord. for ULC of rect. #  2: 1 2
Enter (X,Y) coord. for LRC of rect. #  2: 5 8
Enter (X,Y) coord. for ULC of rect. #  3: 4 6
Enter (X,Y) coord. for LRC of rect. #  3: 8 4
Rect #  0 has area 1.0000
Rect #  1 has area 3.7500
Rect #  3 has area 8.0000
Rect #  2 has area 24.0000
```

分　析

清单 8.2 中的程序包含了头文件 IOSTREAM.H, MATH.H 和 STDIO.H。程序声明了全局常量 MAX_RECT 说明矩形的最多个数。第 11 行包含了结构 point 的声明,它由两个 double 型成员 x 和 y 组成,这是二维点的结构模型。第 16 行包含了矩形模型的结构 rect 的声明,该结构包含了两个 point 类型的成员 ulc 和 lrc, double 型成员 area 和 int 类型成员 id。成员 ulc 和 lrc 代表矩形的左上角和右下角的坐标;成员 area 存储矩形面积;成员 id 存储数字型标识号。

第 23 行中的 typedef 语句定义了有 MAX_RECT 个元素的结构 rect 的数组类型 recArr。函数 main 说明了 rectArr 类型的变量 r, rect 类型的结构 temp 以及 double 类型变量 length 和 width。

main 函数在 31 到 42 行使用了 for 循环提示输入矩形的坐标,计算面积、并对 id 号赋值。第 32、33 以及 35 和 36 行的输出语句分别提示用户输入左上角和右下角的 X、Y 坐标。第 34 和 37 行中的输入语句把坐标分别存入成员 r[i].ulc.x, r[i].ulc.y, r[i].lrc.x 和 r[i].lrc.y。第 38 行的语句把循环控制变量 i 的值存入成员 r[i].id。第 39 行语句用元素 r[i]中的成员 ulc 和 lrc 中的成员 x 计算矩形的长度。第 40 行语句用元素 r[i]中成员 ulc 和 lrc 的成员 y 计算矩形的宽度。第 41 行语句计算矩形面积并存入成员 r[i].area。

第 44 到 51 行的嵌套循环利用成员 area 对数组 r 中的元素排序。循环实现了简单的冒泡排序算法(它对很小的数组很有用)。第 47 行中的 if 语句比较元素 r[i]和 r[j]的所有成员。如果矩形 r[i]的面积比矩形 r[j]的面积大,第 48 到 50 行的语句交换 r[i]和 r[j]的所有成员,交换使用了结构 temp。这一操作表明在一条语句中可以交换一个结构与另一个结构的所有成员。

第 54 和 55 行的 for 循环显示按面积排序后的矩阵。第 55 行的输出语句用 printf 函数显示矩形的 id 号和面积。

8.4　联　　合

联合的存储大小等于其最大成员的存储大小。

新　词

联合(union)
联合是存储互斥成员的特殊结构。

语　法

联合
联合的语法为:

```
union unionTag{
    type1 member1;
    type2 member2;
    ...
    typeN memberN;
};
```

示　例:

```
union Long{
    unsigned mWord[2];
    long mLong;
};
```

联合提供一种进行快速数据转换的简单方法。在过去十年中,昂贵的内存价格以及使用联合合并内存的可行性,使联合一直扮演着重要角色。像结构一样,存取联合成员使用点运算符。

8.5　引 用 变 量

第 2 天中曾学习过在参数类型后放置 & 符号声明引用参数的方法,回想一下,引用参数成了变元的别名。另外,对引用参数进行改变将影响到相应的变元。除了引用参数之外,C++ 还支持引用变量。可以通过使用别名操作引用变量。作为 C++ 程序设计的初学者,一开始时使用引用变量可能会受到很大限制。但另一方面,你可能会越来越频繁地使用引用变量。随着对 C++ 使用的深入,你会发现许多用引用变量进行高级类设计的技巧。本书仅仅讨论引用变量的基本用法。

新　词

引用变量(reference variable)
像引用参数那样,引用变量成为它所存取变量的别名。

语　法

引用变量
声明引用变量的一般语法为:

```
type& refvar;
type& refvar = aVar;
```

refVar 是引用变量,在声明时可以初始化。在使用引用变量之前,必须保证已声明了引用变量并进行了赋值。

示　例:

```
int x = 10, y = 3;
int rx = x;
int& ry;
ry = y; //take the reference
```

下面的例子示意了引用变量的使用。清单 8.3 中列出了程序 REFVAR1.CPP 的源代码。程序用变量本身和它的引用显示并修改了变量的值。程序不需要输入。

键　入　　**清单 8.3　程序 REFVAR1.CPP 的源代码**

```
1: /*
2:    C++ program that demonstrates reference variables
3: */
4:
5: #include <iostream.h>
6:
7: main()
8: {
9:    int x = 10;
10:   int &rx = x;
11:   //display x using x and rx
12:   cout << "x contains "<< x << "\n";
13:   cout << "x contains (using the reference rx)";
14:         << rx << "\n";
15:   // alter x and display its value using rx
16:   x *= 2;
17:   cout << "x contains (using the reference rx)"
18:         << rx << "\n";
19:   // alter rx and display its value using x
20:   rx *= 2;
21:   cout << "x contains "<< x << "\n";
22:   return 0;
23: }
```

以下是清单 8.3 中程序的一个运行范例:

输　出

```
x contains 10
x conatins (using the reference rx) 10
x contains (using the reference rx) 20
x contians 40
```

分　析

清单 8.3 中的程序声明了 int 型变量 x 和 int 型的引用变量 rx。程序初始化变量 x 为 10,引用变量 rx 初始化为 x。

第 12 行的输出语句用变量 x 显示 x 的值;与此相反,第 13 和 14 行的输出语句用引用变量 rx 显示 x 的值。

第 16 行的语句将变量 x 加倍,第 17 和 18 行用引用变量 rx 显示变量 x 的新值。如输出所显示的那样,引用变量真实地显示出变量 x 的更新后的值。

8.6　指 针 概 述

无论是程序还是数据,计算机内存中的每一条信息都存于一明确的地址并占有一定的字节。当程序运行时,变量存在于特定的地址中。对于像 C++ 这样的高级语言,用不着你去关心每个变量的实际地址,它是由编译器和运行系统处理。从概念上讲,程序中的每个变量都是内存地址的标记。使用标记处理数据当然要比使用像 0F64:01AF4 这样的实际的数值地址容易得多。

新　词

地址(address)　　标记(tag)
地址是内存位置。标记是变量的名字。

C++ 与它的上一代语言 C 都是可用于低级系统程序设计的编程语言。实际上,许多程序员把 C 看作是高级汇编。低级系统程序设计要求频繁地操作数据的地址,这就是指针所要扮演的角色。知道一条数据的地址就可以设置和查询它的值。

新　词

指针(pointer)
指针是存储另一变量或信息的地址的特殊变量。

警　告　指针是强有力的语言机构,但如果使用不当也会带来危险,因为它们有可能使系统挂起!当指针正好指向有重要数据或函数的低端内存地址时候就会发生异常。

8.7　指向现有变量的指针

这一节学习如何用指针存取现有变量的值。C++ 要求声明的指针必须与一种数据类型相关联(包括 void)。相关的数据类型可以是预定义类型或用户自定义的结构。

语　法

指针
声明指针的一般语法为:

```
type * pointerName;
type * pointerName = &variable;
```

& 运算符为取地址运算符，用于取得一个变量的地址。

示　例：

```
int * inptr; //pointer to an int
double * realPtr; // pointer to a double
char * aString ;  // pointer to a character
long lv;
long * lp =  &lv;
```

在声明指针的同一行上还可以声明非指针数据：

```
int * intPtr, anInt;
double * realPtr, x;
char * aString, aKey;
```

注　意　C+ + 允许在相关的数据类型的右边置一星号字符，但不能把这种语法解释为相同行上声明的其他各个标识符也都是指针；

```
int*  intPtr; // pointer to an int
double*  realPtr; // pointer to a double
char*  aString; / /pointer to a character
int * intP, j; // intP is a pointer to int, j is an int
double * readPtr, * doublePtr; // both identifiers
                               // are pointers to a double
```

要初始在——像一般变量一样，在使用指针前要初始化。实际上，初始化指针的需要更迫切——使用未初始化的指针将会引起不可预料的麻烦，甚至导致系统挂起！

不要认为未初始化的指针没有危险！

一旦指针包含了变量的地址，便可以用“ * ”运算符后跟以指针名存取变量的值。例如，如果 px 是指向变量 x 的指针，便可以用* px 存取变量 x 的值。

要在指针的左边带上“ * ”运算符存取变量值。

不要忘记使用“ * ”运算符。没有它，语句便成为操作指针中的地址，而不是该地址中的数据。

下面的例子示意了指针的使用。清单 8.4 中列出了程序 PTR1.CPP 的源程序。程序分别用变量自身和它的指针显示并修改变量的值。程序不要求输入

键　入　　清单 8.4　程序 PTR1.CPP 的源代码。

```
1: /*
2:    C+ +  program that demonstrates pointers to existing variables
```

```
 3: * /
 4:
 5: # include < iostream. h>
 6:
 7: main( )
 8: {
 9:    int x =  10;
10:    int*  px =  &x;
11:    / / display x using x and rx
12:    cout < <  "x contains" < <  x < <  "\n";
13:    cout < <  "x contains (using the pointer px)"
14:          < <  * px < <  "\n";
15:    / / alter x and display its value using * px
16:    x * =  2;
17:    cout < <  "x contains (using the reference px)"
18:          < <  * px < <  "\n";
19:    / / alter * px and display value using x
20:    * px * =  2;
21:    cout < <  "x contains " < <  x < <  "\n";
22:    return 0;
23: }
```

以下是清单 8. 4 中程序的一个运行范例:

输　出

```
x contains 10
x contains (using the pointer px ) 10
x contains (using the pointer px ) 20
x contains 40
```

分　析

清单 8. 4 中的程序说明了 int 型变量 x 和 int 型指针 px。程序初始化变量 x 为 10、指针 px 为变量 x 的地址。

第 12 行的输出语句用变量 x 显示 x 的值。与此相反, 第 13 和 14 行输出语句用指针 px 显示变量 x 的值。注意, 语句便用* px 存取变量 x 的值。

第 16 行的语句把变量 x 的值加倍。第 17 和 18 行语句用指针 px 显示变量 x 中的新值。正如输出所显示的, 指针显示了修改后的变量 x 的值。

第 20 行的语句利用指针 px 对变量 x 的值加倍, 注意, 赋值语句在= 运算符的左边使用了* px。第 21 行中的输出语句用变量 x 显示修改后的值。输出再次表明变量 x 与指针 px 是同步的。

8. 8　指向数组的指针

C+ + 和它的上一代语言 C 都支持对数组名的特殊使用。编译器把数组名解释为第一个元素的地址。这样, 如果 x 是一个数组, 表达式 &x[0] 与 x 等价。对于矩阵的情况——

设为 mat——表达式 &mat[0][0]和 mat 等价。一方面，C+ + 和 C 可以像高级汇编语言那样工作，也就是说，一旦有了数据项的地址，便可以得到数值。知道了变量或数组的地址便可以用指针操作它的内容。

新　词

程序变量(program variable)

程序变量是标记内存地址的标号。在程序中使用变量意味着存取变量名所表示的内存单元。在这种意义下，变量成为指向内存单元的名字——指针。

C+ + 允许使用指针存取数组的元素。当存取数组 x 的元素 x[i]时，编译代码完成两项任务：首先获取数组 x 的基地址(也就是第一个数组元素的位置)；其次，用下标 i 计算从数组基地址开始的偏移量。偏移量为 i 乘以数组基本类型的存储大小：

```
address of element x[i] = address of x + i * sizeof(basicType)
```

对于上述等式，假定指针 ptr 的内容为数组 x 的基地址：

```
ptr = x; //pointer ptr points to address of x[0]
```

现在在等式中用 ptr 代替 x，结果如下：

```
addres of element x[i] = ptr + i * sizeof(basicType)
```

为了使 C+ + 和 C 更接近高级汇编，省略了等式中必须要显式声明的数组基本类型的大小。这样，等式写法如下：

```
address of element x[i] = p + i
```

等式说明元素 x[i]的地址为(p + i)。

我们通过下一个程序，即 PTR2.CPP(清单 8.5)来学习用指针存取一维数组。程序是由计算数组中数据平均值的程序 ARRAY1.CPP 修改得到的。程序开始时提示用户输入数据数目和数据自身，然后计算数组中数据的平均值，最后，程序显示平均值。

键　入　　**清单 8.5　程序 PTR2.CPP 的源代码**

```
 1: /*
 2:     C+ + program that demonstrates the use of pointer with
 3:     one-dimension arrays. Program calculates the average
 4:     value of the data found in the array.
 5:*/
 6:
 7: #include <iostream.h>
 8:
 9: const int MAX = 30;
10:
11: main()
12: {
13:
14:     double x[MAX];
15:     // declare pointer and initialize with base
16:     // address of array x
```

```
17:     double * realPtr = x; //same as = &x[0]
18:     double sum, sumx = 0.0, mean;
19:     int n;
20:     // obtain the number of data points
21:     do {
22:          cout << "Enter number of data points [2 to"
23:               << MAX << "] : ";
24:          cin >> n;
25:          cout << "\n";
26:     } while (n < 2 ©©¦n > MAX);
27:
28:     // prompt for the data
29:     for (int i = 0; i < n; i++ ) {
30:          cout << "x[" << i << "] : ";
31:          // use the form * (x+ i) to store data in x[i]
32:          cin >> * (x + i);
33:     }
34:
35:     sum = n;
36:     for (i = 0; i < n; i++ )
37:     // use the form * (realPtr + i) to access x[i]
38:          sumx + = * (realPtr + i);
39:     mean = sumx / sum;
40:     cout << "\nMean = " << mean << "\n\n";
41:     return 0;
42: }
```

以下是清单 8.5 中程序的一个运行范例:

输　出

```
Enter number of data points [2 to 30] : 5
x[0] : 1
x[1] : 2
x[2] : 3
x[3] : 4
x[4] : 5

Mean = 3
```

分　析

清单 8.5 中的程序声明了有 MAX 个元素的 double 型数组 x。另外,还声明了指针 realPtr 并用数组 x 将其初始化。这样,指针 realPtr 存储了数组 x 第一个元素 x[0]的地址。

程序在第 32 行的输入语句中使用了指针表达式* (p + i)。这样,标识符 x 就作为数组 x 的指针而起作用。用表达式* (x + i)访问数组 x 的第 i 个元素,就像表达式 x[i]的作用一样。

程序在第 37 到 38 行的 for 循环中使用了指针 realPtr。表达式* (realPtr + i)等价

于* (x + i), 也等于 x[i]。这样, for 循环使用指针 realPtr 加上偏移量 i, 存取数组 x 的元素。

指针增量/减量的方法

前面的 C+ + 程序在指针 realPtr 中保持了同一地址。利用 for 循环和索引 i 对指针进行算术运算, 可以编写出增加偏移量存取数组 x 的元素的新程序版本。C+ + 提供了另一种方法, 不用明确的偏移值也可以顺序访问数组 x 的元素。这种方法只需对指针使用增量或减量运算符即可, 它也需要用数组基地址初始化指针, 然后用+ + 运算符存取下一个数组元素。以下是上一个程序的修改后的版本, 其中使用了指针增量方法。清单 8.6 中列出了 PTR3.CPP 程序的源代码。

键　入　　清单 8.6　程序 PTR3.CPP 的源代码

```
/*
   C++ program that demonstrates the use of pointers with
   one-dimension arrays. The average value of the array
   is calculated. This program modifies the previous version
   in the following way: the realPtr is used to access the
   array without any help from any loop control variable.
   This is accomplished by 'incrementing' the pointer, and
   consequently incrementing its address. This program
   illustrates pointer arithmetic that alters the pointer's
   address.

*/

#include <iostream.h>

const int MAX = 30;

main ()
{

      double x[MAX];
      double * realPtr = x;
      double sum, sumx = 0.0, mean;
      int i, n;

    do {
        cout << "Enter number of data points [2 to "
             << MAX << "] : ";
        cin >> n;
        cout << "\n";
    } while (n < 2 ©Ð n > MAX);

    // loop variable i is not directly involved in accessing
    // the elements of array x
    for (i = 0; i < n; i++ ) {
        cout << "X[" << i << "] : ";
```

```
37:          // increment pointer realPtr after taking its reference
38:          cin > > * realPtr+ + ;
39:      }
40
41:      // restore original address by using pointer arithmetic
42:      realPtr - = n;
43:      sum = n;
44:      // loop variable i serves as a simple counter
45:      for (i = 0; i < n; i+ + )
46:          // increment pointer realPtr after taking a reference
47:          sumx + = * (realPtr+ + );
48:      mean = sumx / sum;
49:      cout < < "\nMean = " < < mean < < "\n\n";
50:      return 0;
51:
52: }
```

以下是清单 8.6 中程序的一个运行范例:

输　出

```
Enter number of data points [2 to 30] : 5
 x[0] : 10
x[1] : 20
x[2] : 30
x[3] : 40
x[4] : 50

Mean = 30
```

分　析

清单 8.6 中的程序初始化 realPtr 指针为数组 x 的基地址。程序在第 38 行的键盘输入语句中使用了 realPtr 指针。语句用* realPtr+ + 把输入存储到当前存取的数组 x 的元素中, 然后增量指针指向数组 x 的下一个元素。当输入循环结束时, 指针 realPtr 越过了数组 x 的末尾。为了重置指针为数组 x 的基地址, 程序在第 42 行使用了赋值语句。该语句用指针算术运算把指针 realPtr 中的当前地址减去 n 乘以 sizeof(real), 它重置指针 realPtr 的地址以访问数组元素 x[0]。程序在第 47 行的第二个 for 循环中用相同的增量方法计算数据的总和。

8.9　指向结构的指针

C+ + 支持声明和使用指向结构的指针。把结构变量的地址赋值给相同类型的指针使用的是与简单变量指针赋值同样的语法。一旦指针有了结构变量的地址, 便需要用- > 运算符存取结构的成员。

存取结构成员

语　法

用指针存取结构成员的一般语法为:

```
structPtr->aMember
```

示　例:

```
struct point {
  double x;
  double y;
};

point p;
point* ptr = &p;

ptr->x = 23.3;
ptr->y = ptr->x + 12.3;
```

下面是使用指向结构的指针的例子。清单 8.7 中列出了程序 PTR4.CPP 的源代码。这个程序是 STRUCT1.CPP 使用指针的版本。程序提示用户输入四组坐标定义四个矩形,每个矩形由左上角和右下角的 X 和 Y 坐标定义。程序计算每个矩形的面积,并按面积排序,然后依次显示矩形。

键　入

清单 8.7　程序 PTR4.CPP 的源代码

```
 1: /*
 2:    C++ program that demonstrates pointers to structured types
 3: */
 4:
 5: #include<iostream.h>
 6: #include<stdio.h>
 7: #include<math.h>
 8:
 9: const MAX_RECT = 4;
10:
11: struct point {
12:    double x;
13:    double y;
14: };
15:
16: struct rect {
17:    point ulc; // upper left corner
18:    point lrc; // lower right corner
19:    double area;
20:    int id;
21: };
22:
23: typedef rect rectArr[MAX_RECT];
24:
25: main()
26: {
27:    rectArr r;
```

```
28:    rect temp;
29:    rect* pr = r;
30:    rect * pr2;
31:    double length, width;
32:
33:    for (int i = 0; i < MAX_RECT; i++, pr++) {
34:      cout << "Enter (X,Y) corrd. for ULC of rect. # "
35:           << i << ": ";
36:      cin >> pr->ulc.x >> pr->ulc.y;
37:      cout << "Enter (X,Y) coord. for LRC of rect. # "
38:           << i << ": ";
39:      cin >> pr->lrc.x >> pr->lrc.y;
40:      pr->id = i;
41:      length = fabs(pr->ulc.x - pr->lrc.x);
42:      width = fabs(pr->ulc.y - pr->lrc.y);
43:      pr->area = length * width;
44:    }
45:
46:    pr -= MAX_RECT; //reset pointer
47:    // sort the rectangles by areas
48:    for (i = 0; i < (MAX_RECT - 1); i++, pr++) {
49:      pr2 = pr + 1; // reset pointer pr2
50:    for (int j = i + 1; j < MAX_RECT; j++, pr2++)
51:      if (pr->area > pr2->area) {
52:        temp = *pr;
53:        *pr = *pr2;
54:        *pr2 = temp;
55:      }
56:    }
57:
58:    pr -= MAX_RECT - 1; // reset pointer
59:    // display rectangles sorted by area
60:    for (i = 0; i < MAX_RECT; i++, pr++)
61:      printf("Rect # %d has area %5.14lf\n", pr->id, pr->area);
62:    return 0;
63: }
```

以下是清单 8.7 中程序的一个运行范例:

输　出

```
Enter (X,Y) coord. for ULC of rect. # 0: 1 1
Enter (X,Y) coord. for LRC of rect. # 0: 2 2
Enter (X,Y) coord. for ULC of rect. # 1: 1.5 1.5
Enter (X,Y) coord. for LRC of rect. # 1: 3 4
Enter (X,Y) coord. for ULC of rect. # 2: 1 2
Enter (X,Y) coord. for LRC of rect. # 2: 5 8
Enter (X,Y) coord. for ULC of rect. # 3: 4 6
Enter (X,Y) coord. for LRC of rect. # 3: 8 4
Rect # 0 has area 1.0000
Rect # 1 has area 3.7500
Rect # 3 has area 8.000
```

```
Rect #  2 has area 24. 0000
```

分　析

清单 8.7 中的程序在 29 和 30 行分别声明了指针 pr 和 pr2。这些指针访问结构类型 rect。程序用数组 r 的基地址初始化指针 pr。

第 33 行开始的第一个 for 循环用指针 pr 访问数组 r 的元素。循环增量部分包含了表达式 pr++，运用指针算术运算使指针 pr 指向数组 r 中的下一个元素。第 36 和 39 行的输入语句用指针 pr 访问成员 ulc 和 lrc。注意，语句使用了指针访问算符“->”以便指针 pr 访问成员 ulc 和 lrc。第 40 到 43 行的语句也使用指针 pr 和“->”运算符访问成员 id，ulc，lrc 和 area。

第 46 行中的语句将指针中的地址减去 MAX_RECT 个单位(MAX_RECT * sizeof(double)个字节)以重置指针。第 48 到 56 行的嵌套循环使用了指针 pr 和 pr2，外层 for 循环在下一次迭代之前把指针 pr 中的地址增 1。第 49 行语句把 pr + 1 赋值给 pr2，使指针 pr2 初始指向 r 中的元素 i + 1。内层 for 循环在下一次迭代之前把指针 pr2 中的地址增 1。这样，嵌套 for 循环使用指针 pr 和 pr2 访问了数组 r 的元素。第 51 行的 if 语句使用指针 pr 和 pr2 访问成员 area，以比较各矩形的面积。第 52 到 54 行的语句交换指针 pr 和 pr2 所指向的数组 r 的元素。注意，语句用* pr 和* pr2 访问了数组 r 的所有元素。

第 58 行的语句将指针 pr 减去 MAX_RECT - 1 以重置指针中的地址。最后一个 for 循环利用指针 pr 显示数组 r 中的各元素的成员 id 和 area。

程序表明可以仅用指针完成全部数组操作——这种操作是强有力的、也是多种多样的。

8.10　指针和动态内存

迄今为止所提供的程序中变量的空间都是编译时建立的。当程序开始运行时，变量已有了预定义的内存空间。有许多应用程序需要在程序执行时建立新的空间，因此需要为这些运行时的新变量动态分配内存。C++ 设计者选择了首次引入 C 中未出现的新操作符处理动态分配和收回内存的方法。新的 C++ 操作符为 new 和 delete。虽然 C 风格的动态内存函数 malloc，calloc 和 free 仍可使用，最好还是使用操作符 new 和 delete。这些操作符比函数 malloc，calloc 和 free 更重视创建的动态数据的类型。

语　法

new 和 delete 操作符

使用 new 和 delete 操作符创建动态标量变量的一般语法为：

```
pointer = new type;
delete pointer;
```

操作符 new 返回动态分配的变量的地址，操作符 delete 删除指针所指的动态分配的内存。如果操作符 new 动态分配内存失败，返回 NULL(等于 0)指针。因此，在用 new 操作符之后需要测试是否为 NULL 指针，以判定是否遇到麻烦。

示　例：

```
int * pint;
```

```
pint = new int;
* pint = 33;
cout << "Pointer pint stores " << * pint;
delete pint;
```

语　法

动态数组

若要分配和删除动态数组,可使用以下的一般语法:

```
arrayPointer = new type[arraySize];
delete [] arrayPointer;
```

操作符 new 返回动态分配的数组地址。如果分配失败,操作符将指针赋值为 NULL。操作符 delete 删除指针所指的动态分配的数组。

示　例:

```
const int MAX = 10;
int * pint;
pint = new int[MAX];
for (int i = 0; i < MAX; i++)
    * pint [i] = i * i;
for (i = 0; i < MAX; i++)
    cout << * (pint + i) << "\n";
delete [] pint;
```

要一直保持对动态变量和数组的访问。这种访问不需要用来生成这些动态变量和数组的原始指针。下面是一个例子:

```
int* p = new int;
int * q;
* p = 123;
q = p; // q now also points to 123
p = new int; // create another dynamic variable
* p = 345; // p points to 345 whereas q points to 123
cout << * p << " " << * q << " " << (* p + * q) << "\n";
delete p;
delete q;
```

不要忘记当超出动态变量和数组的作用域后要删除它们。

下面的程序,即 PTR5.CPP(清单 8.8)示范了怎样用指针创建和存取动态数据。该程序是经修改计算数组中数据平均值的程序 ARRAY1.CPP 后得到的。程序先提示用户输入实际数据个数并检查合法性,然后要求输入数据并计算数组中数据平均值,最后显示平均值。

键　入　　**清单 8.8　程序 PTR5.CPP 的源代码**

```
1: /*
2:    C++ program that demonstrates the pointers to manage
3:    dynamic data
```

```
 4: * /
 5:
 6: # include < iostream. h>
 7:
 8: const in MAX =  30;
 9:
10: main( )
11: {
12:
13:      double*  x;
14:      double sum, sumx =  0, mean;
15:      int * n;
16:
17:      n =  new int;
18:      if (n = =  NULL)
19:        return 1;
20:
21:      do { // obtain number of data points
22:           cout < <  "Enter number of data points [2 to "
23:           < <  MAX < <  "] : ";
24:           cin > >  * n;
25:           cout < <  "\n";
26:      } while (* n <  2 ©©¦* n >  MAX);
27:      // create tailor- fit dynamic array
28:      x =  new double[* n];
29:      if (! x) {
30:        delete n;
31:        return 1;
32:      }
33:      // prompt user for data
34:      for (int i =  0; i <  * n; i+ + ) {
35:          cout < <  "X[" < <  i < <  "] : ";
36:          cin > >  x[i];
37:      }
38:
39:      //initialize summations
40:      sum =  * n;
41:      // calculate sum of observations
42:      for (i =  0; i <  * n; i+ + )
43:           sumx + =  * (x +  i);
44:
45:      mean =  sumx / sum; // calculate the mean value
46:      cout < <  "\nMean = " < <  mean < <  "\n\n";
47:      // deallocate dynamic memory
48:      delete n;
49:      delete [ ] x;
50:      return 0;
51: }
```

以下是清单 8.8 中程序的一个运行范例:

输　出

```
Enter number of data points [2 to 30] : 5

X[0] : 1
X[1] : 2
X[2] : 3
X[3] : 4
X[4] : 5

Mean = 3
```

分　析

清单 8.8 中的程序用了两个指针用于动态分配。第 13 行说明了第一个指针，用于分配和访问动态数组。第 15 行声明了用于创建动态数组的指针。

第 17 行的语句使用操作符 new 为动态 int 变量分配空间，返回的动态数据的地址存放在指针 n 中。第 18 行的 if 语句判断动态分配是否失败。如果失败，函数 main 退出并返回退出代码 1(标志错误)。

第 21 到 26 行的 do-while 循环提示输入数据点数。第 24 行的语句把输入存放在指针 n 所指的动态变量中。该语句使用指针引用* n 进行访问。while 子句也使用* n 存取动态变量的值。实际上，程序中的所有语句都使用指针引用* n 存取数据的个数。

第 28 行的语句用操作符 new 创建动态数组，该语句用指定的元素数目创建动态 double 型的数组。这一特性显示了用动态分配创建合适的数组的优点。第 29 行的语句判断动态数组分配是否成功。如果失败，第 30 和 31 行删除指针 n 所指的动态变量并以 1 为返回值退出函数。

第 34 到 37 行的 for 循环提示输入动态数组的值。第 36 行的语句把输入存储到动态数组的元素 i 中。注意该语句使用 x[i]存取目标元素，这一形式与静态数组相像。C+ + 对 x[i]的处理与* (x+ i)等同。实际上，程序在第 42 和 43 行的第二个 for 循环中使用了后一种形式。第 43 行的语句使用* (x+ i)的形式存取动态数组元素。

函数 main 的最后一条语句删除动态变量和数组。第 48 行语句删除指针 n 所指动态变量。第 49 行的语句删除指针 x 所指的动态数组。

8.11　远　指　针

在 Intel 80x86 系列处理器的体系结构中都使用了分段式内存，每段 64K。使用段有优点也有缺点。

新　词

近指针(near pointer)　　**远指针**(far pointer)

一般存储模式支持两种指针：近指针和远指针。在段内，可以用近指针存取同一段的数据，指针仅存放段内的偏移地址，因此只需用较少的字节存储地址；与此相反，远指针存放段和偏移地址，需要较多的空间。Windows

应用程序使用远指针。

若要声明远指针，要在指针类型与指针名之间插入关键字 far（有时也用 _far）。

小　　结

今天的课程介绍了用户定义数据类型，包括以下内容：

- 使用 typedef 语句可以创建已有类型的别名并可以定义数组类型。使用 typedef 的语法为：

```
typedef knownType newType;
```

- 用枚举数据类型可以声明唯一的标识符表示逻辑上相关的常量集合。声明枚举类型的语法为：

```
enum enumType {< list enumerated identifiers> };
```

- 用结构可定义按逻辑组织的几个域或成员的新类型。这些成员可以是预定义类型或其他结构。声明结构的语法为：

```
struct structTag{
    < list of members>
};
```

- 联合是结构的变体形式。联合的语法为：

```
union unionTag {
        type1 member1;
        type2 member2;
        . . .
        typeN memberN;
};
```

- 引用变量是所引用的变量的别名。声明引用变量时，在引用变量的数据类型后或变量名的左边放置 &。
- 指针是存放其他变量或数据的地址的变量。C++ 用指针提供灵活高效的对数据和系统资源的管理。
- 指向现有变量的指针使用“&”运算符获取这些变量的地址。有了这些地址，通过指针可访问其关联变量中的数据。用指针访问值时，用“*”后跟以指针名。
- 指针被赋值为数组的基地址后就可以访问数组元素了。C++ 将数组名与基地址等同看待。例如，数组 X 的名字被看作 &X[0]。指针可用于顺序遍历数组元素以存取这些元素的值。
- 指向结构的指针处理结构和存取其成员。C++ 提供了“->”操作符以便指针访问结构成员。
- 用操作符 new 和 delete 可以建立和访问动态数据。用这些操作符可建立动态变量和数组。new 操作符将用于创建和访问数据的指针赋值为动态数据的地址。当信

息不再需要时,delete 操作符协助回收动态数据空间。

· 远指针是存储数据项的段和偏移地址的指针,而近指针仅存储偏移地址。远指针比近指针要求更多的存储单元。

问 与 答

问:C+ + 支持指向预定义类型 void 的指针吗?

答:支持。void* 指针被认为是无类型指针并用于复制数据。

问:因为 C+ + 指针有类型(即使是 void* 指针),是否可以用类型强制把通用的 void * 指针访问的数据转换为非 void* 指针?

答:可以,C+ + 允许对指针引用进行类型强制。例如:

```
void* p = data;
long * lp = (long * )p;
```

指针 lp 用类型强制对访问数据进行了转换。

问:如果用 delete 操作符删除动态数组时后面没有跟空的方括号,会发生什么情况?

答:运行系统仅删除动态数组的第一个元素。其他元素仍保留在内存中,成为不可访问的数据。

问:结构可以包含指向它自己的指针吗?

答:可以,许多动态数据结构的模型使用了这种描述。例如,下述结构是带有基指针链的动态链表的结点模型:

```
struct listNode {
   dataType data;
   listNode * next;
};
```

问:C+ + 允许在声明结构之前声明指向结构的指针类型吗?

答:允许,这一特性使声明动态数据结点成为可能。

问:C+ + 允许指针访问其他指针地址吗?

答:允许,C+ + 支持指向指针的指针(称为双指针)。为声明这种指针,要用两个“ * ”,如下面例子中声明双指针 p:

```
int x;
int * px = &x;
int * * p = &px;
```

表达式* p 访问指针 px,表达式* * p 访问变量 x。

作 业

作业部分所提供的测验题有助于巩固理解所学内容,通过练习可以积累运用所学知识的经验。要尽量弄懂测验和练习的答案再继续下一天的课程。附录 B 的‘答案’中提供

了解答。

测　验

1. 以下语句错在哪里?

```
enum Boolenn { false, true };
enum State { on, off };
enum YesNo { yes, no };
enum DiskDriveStatus { on, off };
```

2. 以下枚举类型的声明是否正确?

```
enum YesNo {no = 0, No = 0, yes = 1, Yes = 1 };
```

3. 下面的程序有什么问题?

```
# include < iostream.h>
main ()
{
  int * p = new int;
  cout < < "Enter a number : " ;
  cin > > * p;
  cout < < "The square of " < < * p < < " = " < < (* p * * p);
  return 0;
}
```

练　习

1. 修改程序 PTR4.CPP,建立程序 PTR6.CPP,使用梳形排序法对矩形的数组排序。
2. 定义可以用于建立整型动态数组模型的结构。结构应有访问动态数据的成员和存储动态数组大小的成员,结构名为 intArrStruct。
3. 定义可以用于建立动态矩阵模型的结构。结构应有访问动态数据的成员以及两个存储行列数的成员,结构名为 matStruct。

第9天

字 符 串

在第 1 天到第 8 天所举的例子大多是针对数值计算的,少数是面向字符处理的。读者可能已经对所有这些例子中都缺少字符串渐生疑问。今天这一课就是讨论 C+ + 中的串处理的,主要内容如下:

- C+ + 中的串
- 串输入
- 使用标准串函数库
- 串赋值
- 取得串长度
- 串连接
- 串比较
- 串转换
- 串逆置
- 字符定位
- 子串查找

9.1 C+ + 字符串

C+ + (和它的上一代语言 C)没有预定义的串类型。像 C 一样,C+ + 将字符串看作以 ASCII 值为 0 的空字符(‘\0’)结尾的字符数组。

新 词

空终结符(null terminator) ASCIIZ 串(ASCIIZ string)

字符 ‘\0’也称为空终结符。以空终结符结尾的字符串有时称为 ASCIIZ 串,其中字母 Z 即代表 0,也就是空终结符的 ASCII 码。

所有的串中都必须有空终结符,并且在计算串长时应将该符计入在内。当声明一字符串变量为字符数组时,应保证为空终结符保留一个额外空间。用空终结符的优点在于可以建立不受任何 C+ + 的具体实现而限制的字符串。另外,ASCIIZ 串的结构也很简单。

注 意 第 8 天的课程讨论了指针如何访问和操作数组元素。在对串中字符的操作上,C 和 C+ + 广泛使用了这一编程特性。

要包括空终结符的额外空间——当指定字符串的存储大小时。

不要声明串变量为单个字符的数组,这样的变量是无用的!

9.2 串　输　入

迄今为止所给出的程序中在输出流语句里显示串内容。因此,C+ + 支持将字符串作为非预定义的数据类型的特殊情况进行流输出(可以说这一支持来自于大众的要求!)。用串变量进行串输出时用的是同样的操作符和语法。对于串输入,因为串中经常包含被插入操作符忽略的空格,所以插入操作符不能工作得很好。可以用 getline 函数代替插入操作符,该函数读取指定数目的字符。

语　法

getline 函数

重载函数 getline 的语法为:

```
istream& getline(signed char* buffer,
                 int size,
                 char delimiter = '\n');
istream& getline(unsigned char* buffer,
                 int size,
                 char delimiter = '\n');
```

参数 buffer 是指向从流中接收字符的串的指针;参数 size 指定读取的最大字符数;参数 delimiter 指定了定界符,它可以使串输入在达到参数 size 指定的字符数之前停止。参数 delimiter 的缺省值为'\n'。

示　例:

```
# include <iostream.h>
main()
{
char name[80];
cout << "Enter your name: "
cin.getline(name, sizeof(name) - 1);
cout << "Hello "<< name << ", how are you";
return 0;
}
```

9.3 使用 STRING.H 函数库

C 程序员们开发了标准的串函数库 STRING.H,使其包含了最常用的串操作函数。STDIO.H 和 IOSTREAM.H 头文件中的函数原型也支持串 I/O。各个 C+ + 编译器生产商也开发了 C+ + 式样的串函数库。这些库用类建立串模型(第 11 天将讲述类)。但是,这些串函数库是非标准的,而 STRING.H 中的 C 式样的串例程是 ANSI C 标准的一部分。

在下面的几个小节中将给出几个其原型在 STRING.H 头文件中的函数，但它们只是其中一部分，而不是全部。

STRING.H 中有些串函数的版本不止一个。冠以字符“_f”、“f”或“_”的附加版本可操作用远指针访问的串。

9.4　串　赋　值

C++ 支持两种串赋值方法。在初始化时可以将串文字赋值给串变量。这种方法较简单，它用到= 运算符和赋值的串。

语　法

初始化串

初始化串的语法为：

```
char stringVav[stringSize] = stringLiteral;
```

示　例：

```
char aString[81] = "Visual C++ in 21 days";
char name[] = "Namir Shammas";
```

将一个 ASCIIZ 串赋值给另一个串的第 2 种方法是使用函数 strcpy。函数假定被拷贝的串以空字符结尾。

语　法

strcpy 函数

函数 strcpy 的原型是：

```
char * strcpy(char * target, const char * source)
```

函数把源串中的字符拷贝到目标串中，并假定目标中有足够的空间容纳源串。

示　例：

```
char name[41];
strcat(name, "Visual C++ ");
```

变量 name 中含有串 "Visual C++ "。

用函数 strdup 可以将字符复制到另一串中，但要为目标串分配所需的空间。

语　法

strdup 函数

函数 strdup 的原型是：

```
char * strdup(const char * source)
```

该函数复制源串中的字符，并返回指向复制串的指针。

示　例：

```
char* string1 = "The reign in Spain";
char* string2;
string2 = strdup(string1);
```

上例中，在为 string2 分配内存空间之后，将 string1 的内容复制到 string2 中。

串函数库还提供了 strncpy，以支持从一个串中拷贝指定数目的字符到另一个串。

语　法

strncpy 函数

函数 strncpy 的原型是：

```
char * strncpy(char* target, const char* source, size-t num);
```

该函数从原串中拷贝 num 个字符到目标串中。如果需要，函数执行字符的截断和填充。

示　例：

```
char str1[] = "Pascal";
char str2[] = "Hello there";

strncpy(str1, str2, 6);
```

变量 str1 中现在包含了串 "Hello"。

注　意　许多初学 C++ 的程序员对用指针管理串是很陌生的。实际上，通过把要操作的串的第一个字符的地址赋值给指针，便可以操作串尾部分。例如，声明串 str1 如下：

```
char str1[41] = "Hello World";
char str2[41];
char* p = str1;
lp += 6; // p now points to substring "World" in str
strcpy(str2, p);
cout << str2 << "\n";
```

输出语句显示串 "World"。这个例子示意了如何用指针实现字符的位置偏移。

9.5　串的长度

许多串操作需要求出串中的字符数。STRING.H 库中提供了函数 strlen 以返回字符的个数，但不计入串中的空终结符(NULL)。

语　法

strlen 函数

函数 strlen 的原型是：

```
size-t strlen(const char * string)
```

函数 strlen 返回参数 string 中的字符数。结果类型 size-t 代表一般的整数类型。

示　例：

```
char str[] = "1234567890";
unsigned i;
i = strlen(str);
```

这些语句将变量 i 赋值为 10。

9.6　串的连接

我们经常需要将两个或更多的串连接起来以产生另一个串。函数 strcat 允许把一个串连接到另一个串上。

新　词

连接(concatenate)
连接串就是把它们合并或链接到一起。

语　法

strcat 函数

```
char * strcat(char * target, const char* source)
```

该函数把源串中的内容添加到目标串中,并返回指向目标串的指针。函数假定目标串能够容纳源串中的字符。

示　例:

```
char string[81];
strcpy(string, "Visual ");
strcat(string, "C++ ");
```

变量 string 中现在包含了"Visual C++ "。

函数 strncat 将源串中指定数目的字符连接到目标串中。

语　法

strncat 函数
函数 strncat 的原型是:

```
char * strncat(char* target, const char* source, size_t num)
```

函数把源串中 num 个字符连接到目标串,并返回指向目标串的指针。

示　例:

```
char str1[81] = "Hello I am ";
char str2[41] = "Thomas Jones";
strncat(str1, str2, 6);
```

变量 str1 包含了"Hello I am Thomas"。

要用函数 strncat 控制连接的字符数——当不能确定目标串的容量时。

不要假设目标串总有足够的空间存储源串的字符。

我们来看一个使用 getline, strlen 和 strcat 函数的程序。清单 9.1 中列出了程序 STRING1.CPP 的源代码。程序完成下列任务:

- 提示输入第一个串(输入不应超过 40 个字符);
- 提示输入第二个串(输入不应超过 40 个字符);

· 显示输入的各串的字符数;
· 把第二个串连接到第一个串;
· 显示连接串中的字符数;
· 提示输入要查找的字符;
· 提示输入替换字符;
· 显示将所有查找到的字符用替换字符转换后的连接串。

键　入　　清单 9.1　程序 STRING1.CPP 的源代码

```
1: /*
2:    C++ program that demonstrates C-strle strings
3: */
4:
5: #include <iostream.h>
6: #include <string.h>
7:
8: const unsigned MAX1 = 40;
9: const unsigned MAX2 = 80;
10:
11: main()
12: {
13:
14:       char smallStr[MAX1+1];
15:       char bigStr[MAX2+1];
16:       char findChar, replChar;
17:
18:       cout << "Enter first string:\n";
19:       cin.getline(bigStr, MAX2);
20:       cout << "Enter second string:\n";
21:       cin.getline(smallStr, MAX1);
22:       cout << "String 1 has " << strlen(bigStr)
23:            << "characters\n";
24:       cout << "String 2 has " << strlen(smallStr)
25:            << "characters\n";
26:       // concatenate bigStr to smallStr
27:       strcat(bigStr, smallStr);
28:       cout << "Concatenated strings are:\n"
29:            << bigStr << "\n";
30:       cout << "New string has " << strlen(bigStr)
31:            << "characters\n";
32:       // get the search and replacement characters
33:       cout << "Enter search character: ";
34:       cin >> findChar;
35:       cout << Enter replacement character: ";
36:       cin >> replChar;
37:       // replace characters in string bigStr
38:       for(unsigned i = 0; i < strlen(bigStr); i++)
39:         if (bigStr[i] == findChar)
40:            bigStr[i] = replChar;
41:       // display the updated string bigStr
```

```
42:       cout < <  "New string is: \n"
43:             < <  bigStr;
44:       return 0;
45: }
```

以下是清单 9. 1 中程序的一个运行范例:

输　出

```
Enter first string:
The rain in Spain stays
Enter second string:
 mainly in the plain
String 1 has 23 characters
String 2 has 20 characters
Concatenated strings are:
The rain in Spain stays mainly in the plain
New string has 43 characters
Enter search character : a
Enter replacememt character : A
New string is:
The rAin in SpAin stAys mAinly in the plAin
```

分　析

清单 9. 1 中的程序包含了串操作函数的头文件 STRING. H。第 8 和第 9 行声明了分别用于定义小串和大串长度的全局常量 MAX1 和 MAX2。函数 main 声明了两个串: smallStr 和 bigStr。第 14 行声明了存储 MAX1+ 1 个字符(空字符要多占一个空间)的变量 smallStr。第 15 行声明了存储 MAX2+ 1 个字符的变量 bigStr。第 16 行声明了 char 类型变量 findChar 和 replChar。

第 18 行的输出语句提示用户输入第一个串。第 19 行的语句用流输入函数 getline 读取输入并存到变量 bigStr 中。函数调用指明至多可输入 MAX2 个字符。第 20 行的输出语句提示输入第二个串。第 21 行的语句用流输入函数 getline 读取输入并存入变量 smallStr 中。函数调用指明至多可输入 MAX1 个字符。

第 22 到 25 行的输出语句显示变量 bigStr 和 smallStr 中的字符数。每一输出语句都调用了函数 strlen 并以串变量为参数。

第 27 行的语句把变量 smallStr 中的串连接到变量 bigStr 上。第 28 和 29 行的输出语句显示更新的串 bigStr。第 30 和 31 行的输出语句显示更新后的串变量 bigStr 中的字符数, 该语句也使用了函数 Strlen 获得字符数。

第 33 行的语句提示输入查找字符。第 34 行的语句读取输入并存入变量 findChar 中。第 35 行的语句提示输入替换字符。第 36 行的语句读取输入并存入变量 replChar。

第 38 到 40 行的 for 循环转换串 bigStr 中的字符。该循环使用控制变量 i 并以 1 为增量从 0 到 strlen(bigStr)- 1 进行迭代。第 39 行的语句判定 bigStr 中第 i 个字符是否与变量 findChar 中的字符匹配。如果条件为真, 程序执行第 40 行的语句, 把变量 replChar 中

的字符赋值给变量 bigStr 中的第 i 个字符。循环表明如何通过访问串中的每一个字符操作串的内容。

第 42 和 43 行的输出语句显示更新后的串 bigStr。

9.7 串　比　较

因为串是字符数组，STRING.H 函数库提供了一组串比较函数。函数根据字符的 ASCII 值比较两个串中的字符。这些函数有 strcmp，stricmp，strncmp 和 strnicmp。

函数 strcmp 对两个串中所有可能的字符进行区分大小写的比较。

语　法

strcmp 函数

函数 strcmp 的原型是：

```
int strcmp(const char * str1, const char * str2);
```

函数比较串 str1 和 str2。整数返回值表明比较的结果：

< 0　当 str1 小于 str2 时

= 0　当 str1 等于 str2 时

> 0　当 str1 大于 str2 时

示　例：

```
char string1[] = "Visual C++";
char string2[] = "Visual Basic";
int i;

i = strcmp(string1, string2);
```

最后一条语句将一个负数赋值给变量 i，因为变量 string1 中的字符串小于变量 string2 中的串。

函数 stricmp 对两个串中所有可能的字符进行不区分大小写的比较。

语　法

stricmp 函数

函数 stricmp 的原型是：

```
int stricmp(const char * str1, const char * str2);
```

函数比较串 str1 和 str2，比较时不区分大小写字符。整数返回值表明了比较结果：

< 0　当 str1 小于 str2 时

= 0　当 str1 等于 str2 时

> 0　当 str1 大于 str2 时

示　例：

```
char string1[] = "Visual C++";
char string2[] = "VISUAL C++";
int i;
```

```
i = stricmp(string1, string2);
```

最后一条语句给变量 i 赋值 0, 因为变量 string1 和 string 中的串仅在大小写上不同。

函数 strncmp 在两个串首指定长度上执行区分大小写的比较。

语　法

strncmp 函数

函数 strncmp 的原型是:

```
int strncmp(const char * str1, const char * str2, size_t num);
```

函数比较两个串 str1 和 str2 中的前 num 个字符。整数返回值表明比较结果:

< 0　当 str1 小于 str2 时

= 0　当 str1 等于 str2 时

> 0　当 str1 大于 str2 时

示　例:

```
char string1[] = "Visual Basic";
char string2[] = "Visual C++ ";
int i;

i = strncmp(string1, string2, 8);
```

因为"Visual B"小于"Viusual C",所以将一个负数赋值给变量 i。

函数 strnicmp 在两个串首指定长度上进行不区分大小写的比较。

语　法

strnicmp 函数

函数 strnicmp 的原型是:

```
int strnicmp(const char * str1, const char * str2, size_t num);
```

函数比较两个串 str1 和 str2 中的前 num 个字符, 不区分字符的大小写。整数返回值表明了比较的结果:

< 0　当 str1 小于 str2 时

= 0　当 str1 等于 str2 时

> 0　当 str1 大于 str2 时

示　例:

```
char string1[] = "Visual C++ ";
char string2[] = "VISUAL Basic"
int i;

i = strnicmp(string1, string2, 6);
```

因为串"Visual "和"VISUAL "仅在字符的大小写上不同, 所以变量 i 将赋值为 0。

我们来看一个串比较的例子。清单 9.2 中的程序建立串数组并用数据做了初始化, 然后显示无序的串数组, 进行排序后, 再显示有序数组。

键　入　　清单 9.2　程序 STRING2.CPP 的源代码

```
1: /*
2:    C++ program that demonstrates comparing strings
3: */
4:
5: #include <iostream.h>
6: #include <string.h>
7:
8: const unsigned STR_SIZE = 40;
9: const unsigned ARRAY_SIZE = 11;
10: const int TRUE = 1;
11: const int FALSE = 0;
12:
13: main()
14: {
15:
16:     char strArr[STR_SIZE][ARRAY_SIZE] =
17:         { "California", "Virginia", "Alaska", "New York",
18:           "Michigan", "Nevada", "Ohio", "Florida",
19:           "Washington", "Oregon", "Arizona"};
20:     char temp[STR_SIZE];
21:     unsigned n = ARRAY_SIZE;
22:     unsigned offset;
23:     int inOrder;
24:
25:     cout << "Unordered array of strings is: \n";
26:     for (unsigned i = 0; i < ARRAY_SIZE; i++)
27:     cout << strArr[i] << "\n";
28:
29:     cout << "\nEnter a non-space character and press Enter";
30:     cin >> temp[0];
31:     cout << "\n";
32:
33:     offset = n;
34:     do {
35:       offset = (8 * offset) / 11;
36:       offset = (offset == 0) ? 1 : offset;
37:       inOrder = TRUE;
38:       for (unsigned i = 0, j = offset;
39:            i < n - offset; i++, j++)
40:        if (strcmp(strArr[i], strArr[j]) > 0) {
41:          strcpy(temp, strArr[i]);
42:          strcpy(strArr[i], strArr[j]);
43:          strcpy(strArr[j], temp);
44:          inOrder= FALSE;
45:        }
46:    } while (! (offset == 1 && inOrder));
47:
48:    cout << "Sorted array of strings is:\n";
49:    for (i = 0; i < ARRAY_SIZE; i++)
50:    cout << strArr[i] << "\n";
```

```
51:     return 0;
52: }
```

以下是清单 9.2 中程序的一个运行范例:

输　出

```
Unordered array of strings is:
California
Virginia
Alaska
New York
Michigan
Nevada
Ohio
Florida
Washington
Oregon
Arizona
Enter a non-space character and press Enter
Sorted array of strings is:
Alaska
Arizona
California
Florida
Michigan
Nevada
New York
Ohio
Oregon
Virginia
Washington
```

分　析

清单 9.2 程序在第 8 到 11 行声明了全局常量 STR_SIZE, ARRAY_SIZE, TRUE, 和 FALSE。常量 STR_SIZE 指定每个串的长度。常量 ARRAY_SIZE 表明程序所用数组中串的数目。常量 TRUE 和 FALSE 代表了串数组排序中用到的布尔值。函数 main 声明了数组 strArr(实际上, 变量 StrArr 为字符矩阵), 它有 ARRA_SIZE 个字符。注意, 声明中的第一维描述了每个串的长度, 第二维描述了数组的大小。函数还初始化了数组 StrArr, 声明了用作交换缓冲区的变量 temp。第 21 到 23 行声明了杂用变量。

第 25 行的语句在显示无序数组 StrArr 的元素之前显示了标题。第 26 到 27 行的 for 循环显示这些元素。循环使用控制变量 i 以 1 为增量从 0 到 ARRAY_SIZE - 1 进行迭代。第 27 行的输出语句用表达式 StrArr[i]显示字符串的 i 号元素。

第 29 到 30 行的输出和输入语句提示输入一个非空格字符。在程序未对数组排序和显示有序元素之前, 该输入检测无序数组。

第 33 到 46 行的语句用组合排序法排序。注意, 第 40 行的 if 语句用函数 strcmp 比较

元素 i 和 j, 分别利用表达式 strArr[i]和 strArr[j]访问。第 41 到 43 行中的语句用函数 strcpy 和交换缓冲区 temp 交换元素 i 和 j。

在显示有序的数组元素之前, 第 48 行的输出语句显示了标题。第 49 和 50 行的 for 循环显示这些元素。循环使用控制变量 i 以 1 为增量从 0 到 ARRAY_SIZE- 1 进行迭代。第 50 行的输出语句用表达式 strArr[i]显示串的 i 号元素。

9.8 串的转换

STRING.H 函数库提供了函数_strlwr 和_strupr 分别把串中的字符转换为大写或小写。注意在 C 的书中这些函数更一般的调用是 strlwr 和 strupr(没有前导下划线字符)。

语　法

_strlwr 函数

函数_strlwr 的原型是:

```
char * _strlwr(char * source)
```

该函数将 source 串中的大写字符转换为小写字符, 对其他字符没有影响。函数返回指向 source 的指针。

示　例:

```
char str[]= "HELLO THERE";

_strlwr(str);
```

变量 str 现在包含了串"hello there"。

语　法

_strupr 函数

函数_strupr 的原型是:

```
char * _strupr(char * source)
```

函数将 source 串中的小写字符转换为大写字符, 其它字符没有影响。函数返回指向串 source 的指针。

示　例:

```
char str[]= "Visual C++ ";

_strupr(str);
```

变量 str 现在包含了串"VISUAL C++ "。

要留副本——如果后来程序中要用原来的变元, 对函数_strlwr 和_strupr 的变元要留副本。

不要假设对同一个变量使用函数_strlwr 后再用函数_strupr(或者反过来)就可恢复变量中原来的字符。

9.9　串　逆　置

STRING.H 函数库提供了函数 strrev 以逆置串中的字符。

语　法

strrev 函数

函数 strrev 的原型是:

```
char *  strrev (char *  str)
```

此函数将串 str 中字符的顺序反转并返回指向串 str 的指针。

示　例:

```
char string[ ]= "Hello";
strrev(string);
cout < <  string;
```

这将显示 "olleH"。

我们来看一个处理串中字符的例子。清单 9.3 中列出了程序 STRING3.CPP 的源代码。程序完成以下任务:

- 提示输入一个串;
- 显示输入内容;
- 显示输入的小写版本;
- 显示输入的大写版本;
- 显示键入字符的逆序;
- 如果输入中无大写字符,显示相应消息
- 如果输入中无小写字符,显示相应消息;
- 如果输入中字符对称,显示相应消息。

键　入　　清单 9.3　程序 STRNG3.CPP 的源代码

```
 1: /*
 2:    C+ +  program that demonstrates manipulating the
 3:    characters in a string
 4: *  /
 5:
 6: #  include < iostream.h>
 7: #  include < string.h>
 8:
 9: const unsigned STR. SIZE =  40;
10: const int TRUE =  1;
11: const int FALSE =  0;
12:
13: main( )
14: {
```

```
15:     char str1[STR_SIZE+1];
16:     char str2[STR_SIZE+1];
17:     int isLowerCase;
18:     int isUpperCase;
19:     int isSymmetrical;
20:
21:
22:     cout << "Enter a string: ";
23:     cin.getline(str1, STR_SIZE);
24:     cout << "Input: " << str1 << "\n"
25:     // copy str1 to str2
26:     strcpy(str2, str1);
27:     // convert to lowercase
28:     strlwr(str2);
29:     isLowerCase = (strcmp(str1, str2) == 0) ? TRUE : FALSE;
30:     cout << "Lowercase: " << str2 << "\n";
31:     // convert to uppercase
32:     strupr(str2);
33:     isUpperCase = (strcmp(str1, str2) == 0) ? TRUE : FALSE;
34:     cout << "Uppercase: " << str2 << "\n";
35:     // copy str1 to str2
36:     strcpy(str2, str1);
37:     // reverse characters
38:     strrev(str2);
39:     isSymmetrical = (strcmp(str1, str2) == 0) ? TRUE : FALSE;
40:     cout << "Reversed: " << "\n";
41:     if (isLowerCase)
42:     cout << "Your input has no uppercase letters\n";
43:     if (isUpperCase)
44:       cout << "Your input has no lowercase letters\n";
45:      if (isSymmetrical)
46:      cout << "Your input has symmetrical characters\n";
47:      return 0;
48: }
```

以下是清单 9.3 中程序的一个运行范例:

输　出

```
Enter a string: level
Input: level
Lowercase: level
Uppercase: LEVEL
Reversed: level
Your input has no uppercase letters
Your input has symmetrical characters
```

分　析

清单 9.3 中的程序在函数 main 中声明了串变量 str1 和 str2。每个串存储 STR_SIZE + 1 个字符(包括空终结符)。函数还声明了标志 isLowerCase, isUpperCase 和 isSymmetrical。第 22 行的输出语句提示输入一个串。第 23 行的语句使用串输入函数 getline 把输入存到变量 str1 中。第 24 的语句回显输入。

第 26 行的语句把变量 str1 中的字符拷贝到变量 str2 中。第 28 行的语句调用函数 strlwr 转换变量 str2 中的字符。程序对变量 str2 中的字符操作, 而变量 str1 中的原输入不变。第 29 行的语句调用函数 strcmp 比较 str1 中的字符。仅在输入中无大写字母时两个串相等。如果上述条件为真, 则把常量 TRUE 赋值给标志 isLowerCase; 否则, 语句将 FALSE 赋值给标志 isLowerCase。第 30 行的输出语句显示变量 str2 中的字符。

第 32 行的语句用变元 str2 调用函数 strupr。函数调用把 str2 中的任一小写字母转换为大写字母。第 33 行的语句用函数 strcmp 比较 str1 与 str2 中的字符, 仅当输入中无小写字母时两个串相等。如果上述条件为真, 则把常量 TRUE 赋值给 isUpperCase; 否则, 标志 isUpperCase 赋值为 FALSE。第 34 行的输出语句显示变量 str2 中的字符。

为了逆序显示原输入字符, 程序再次调用函数 strcpy 拷贝变量 str1 中的字符到 str2。第 38 行的语句以 str2 为变元调用函数 strrev。第 39 行的语句调用函数 strcmp 比较 str1 与 str2 中的字符。仅当输入为对称字符时两个串相等。如果上述条件为真, 则把标志 isSymmetrical 赋值为常量 TRUE; 否则, 标志 isSymmetrial 赋值为 FALSE。第 40 行的输出语句显示变量 str2 的字符。

程序在第 41, 43, 45 行使用 if 语句指明输入为特殊字符。当变量 isLowerCase 为 TRUE 时, 第 41 行的 if 语句显示输入中无大写字母的消息。当变量 isUpperCase 为 TRUE 时, 第 43 行的 if 语句显示输入中无小写字母的消息。第 45 行的 if 语句在变量 isSymmetrial 为 TRUE 时, 显示输入不是对称字符。

9.10　字 符 定 位

STRING.H 函数库提供了一些用于定位串中字符的函数。这些函数包括 strchr, strrchr, strspn, strcspn 和 strpbrk。这些函数允许在串中查找字符或简单的字符模板。

函数 strchr 定位字符在串中第一次出现的位置。

语　法

strchr 函数

函数 strchr 的原型是:

```
char* strchr(const char * target, int c)
```

此函数定位模板 c 在串 target 中第一次出现的位置。函实的返回指针指向串 target 中与指定模板 c 相匹配的字符。如果字符 c 未在串 target 中出现, 函数返回 NULL。

示　例:

```
char str[81] = "Visual C++";
```

```
char * strPtr;
strPtr = strchr(str, '+');
```

指针 StrPtr 指向串 str 的子串"+ + "。

函数 strrchr 定位一字符在串中出现的最后位置。

语　法

strrchr 函数

函数 strrchr 的原型为:

```
char * strrchr (const char * target, int c)
```

此函数定位串 target 中最后一次出现模板 c 的位置。函数的返回指针指向串 target 中与指定模板 c 匹配的字符。如果字符 c 不在串 target 中出现,函数返回 NULL。

示　例:

```
char str[81] = "Visual C++ is here";
char * strPtr;

strPtr = strrchr(str, '+');
```

指针 strPtr 指向串 str 的子串"+ is here"。

函数 strspn 返回字符串中与字符模板的任意字符匹配的前导部分的字符个数。

语　法

函数 strspn

函数 strspn 的原型为:

```
size_t strspn(const char* target, const char* pattern)
```

函数 strspn 返回串 target 中与串 pattern 的任意字符匹配的前导部分的字符数。

示　例:

```
char str[] = "Visual C++ 1.0";
char substr[] = "Vailus ";
int index;

index = strspn(str, substr);
```

该语句将变量 index 赋值为 7,因为 str 的前 7 个字符中的每一个在 substr 中都有相匹配的字符。

函数 strcspn 扫描串,并返回完全不出现子串的前导部分的字符数。

语　法

strcspn 函数

函数 strcspn 的原型为:

```
size_t strcspn(const char * str1, const char * str2)
```

该函数扫描 str1 并返回完全不出现子串 str2 中字符的最左子串的长度。

示　例:

```
char strng[] = "The rain in Spain";
int i;
```

```
i = strcspn(strng, "in");
```

此例将变量 i 赋值为 8("The rain "的长度)。

函数 strpbrk 在串中寻找模板字符中的任一字符第一次出现的位置。

语　法

strpbrk 函数

函数 strpbrk 的原型为:

```
char * strpbrk(const char * target, const char * pattern)
```

函数在 target 串中寻找第一次出现串 pattern 中任一字符的位置。如果 pattern 中的字符不在串 target 中出现,函数返回 NULL。

示　例:

```
char * str = "Hello there how are you";
char * substr = "hr";
char * ptr;
ptr = strpbrk(str, substr);
cout << ptr << "\n";
```

这将显示"here how are you",因为先遇到'h'后遇到'r'。

9.11　串　定　位

STRING.H 函数库提供了函数 strstr,用它可以在串中查找子串。

语　法

strstr 函数

函数 strstr 的原型为:

```
char * strstr(const char * str, const char * substr)
```

函数扫描串 str,查找第一次出现 substr 的位置。函数返回指向串 str 中与参数 substr 匹配的字符的指针。如果串 substr 不在串 str 中出现,函数返回 NULL。

示　例:

```
char str[] = "Hello there! how are you";
char substr[] = "how";
char * ptr;
ptr = strstr(str, substr);
cout << ptr << "\n";
```

因为串中找到了匹配的"how",所以显示"how are you"。指针 ptr 指向原串以'how'开始的剩余部分。

要在调用函数 strstr 之前使用函数 strrev 如果想寻找串最后一次出现的位置。
不要忘记对主串与查找串都求逆序,使用 strrev 函数定位查找串最后出现的位置时。

串函数库还提供了函数 strtok，它可以依据指定的定界符集把串分割为子串。

语　法

标记(token)

子串有时也称为标记。

语　法

strtok 函数

函数 strtok 的原型为：

```
char * strtok(char * target, const char * delimiters)
```

函数在目标串中寻找标记。由一个字符串提供了一组分隔符。下述例子显示了函数如何返回串中的标记。在找到每一个标记后函数通过插入 \0 修改目标串。要确保把原目标串拷贝到另一个串变量中保存起来。

示　例：

```
# include < stdio.h>
# include < string.h>

main()
{
   char * str = "(Base- Cost + Profit) * Margin";
   char * tkn = "+ * ()";
   char * ptr = str;

   printf("%s\n",str);
   // the first call looks normal
   ptr = strtok(str, tkn);
   printf("\n\nis broken into: %s", ptr);
   while (ptr) {
        printf(", %s", ptr);
        // must make first argument a NULL character
        ptr = strtok(NULL, tkn);
   }
   printf("\n\n");
}
```

程序运行时显示如下：

```
(Base- Cost+ Profit) * Margin
```

分解为 Base- Cost，Profit，Margin。

要记住提供 NULL 作为第一个参量，当函数 strtok 查找下一标记时。

不要忘了把函数 strtok 中的目标串赋值一个拷贝。

请看一个查找字符和串的例子。清单 9.4 中列出了程序 STRING4.CPP 的源代码。该程序完成以下任务：

- 提示输入主串；
- 提示输入查找串；

- 提示输入查找字符;
- 显示字符标尺和主串;
- 显示查找字符在主串中出现处的索引。

键　入　　清单 9.4　程序 STRNG4.CPP 的源代码

```
1: /*
2:    C++ program that demonstrates searching for the
3:    characters and strings
4: */
5:
6: #include <iostream.h>
7: #include <string.h>
8:
9: const unsigned STR_SIZE = 40;
10:
11: main()
12: {
13:       char mainStr[STR_SIZE+1];
14:       char subStr[STR_SIZE+1];
15:       char findChar;
16:       char *p;
17:       int index;
18:       int count;
19:
20:       cout << "Enter a string: ";
21:       cin.getline(mainStr, STR_SIZE);
22:       cout << "Enter a search string: ";
23:       cin.getline(subStr, STR_SIZE);
24:       cout << "Enter a search character: ";
25:       cin >> findChar;
26:
27:       cout << "          1         2         3         4\n";
28:       cout << "01234567890123456789012345678901234567890\n";
29:       cout << mainStr << "\n";
30:       cout << "Searching for string " << subStr << "\n";
31:       p = strstr(mainStr, subStr);
32:       count = 0;
33:       while (p) {
34:         count++;
35:         index = p - mainStr;
36:         cout << "Match at index " << index << "\n";
37:         p = strstr(++p, subStr);
38:       }
39:       if (count == 0)
40:         cout << "No match for substring in main string\n";
41:
42:       cout << "Searching for character " << findChar << "\n";
43:       p = strchr(mainStr, findChar);
44:       count = 0;
45:       while (p) {
```

```
46:         count++;
47:         index = p - mainStr;
48:         cout << "Match at index " << index << "\n";
49:         p = strchr(++p, findChar);
50:       }
51:       if (count == 0)
52:         cout << "No match for search character in main string\n";
53:       return 0;
54: }
```

以下是清单 9.4 中程序的一个运行范例:

输　出

```
Enter a string : there, there, and everywhere
Enter a search string : there
Enter a search character : e
1          2          3          4
012456789012345678901234567890123456789O
here, there, and everywhere
Searching for string here
Match at index 0
Match at index 7
Match at index 23
Searching for character e
Match at index 1
Match at index 3
Match at index 8
Match at index 10
Match at index 17
Match at index 19
Match at index 24
Match at index 26
```

分　析

清单 9.4 中的程序声明了串 mainStr 和 subStr, 它们分别代表主串和查找串。程序还声明了变量 findChar 以存储要查找的字符。此外, 程序还声明了字符指针 p 和 int 类型度量 index 和 count。

第 20 行的输出语句提示输入字符串。第 21 行的语句调用流输入函数 getLine, 将输入保存到变量 mainStr 中。第 22 行中的输出语句提示输入查找串。第 23 行中调用流输入函数 getline, 将输入存入变量 substr。第 24 行的语句提示输入查找字符。第 25 行的语句读取输入并存入变量 findChar。

第 27 到 29 行的输出语句显示标尺以及在标尺下对准的输入。第 30 行的语句告知用户, 程序正在查找输入的子串。查找从第 31 行的语句开始, 该语句调用函数 strstr 确定串 subStr 在串 mainStr 中第一次出现的位置。第 32 行语句对变量 count 赋值 0, 用于记录串

mainStr 包含串 subStr 的次数。

程序在第 33 到 38 行用 while 循环确定所有 subStr 在 mainStr 中的位置。while 循环的条件检测指针 p 的地址，如果指针不为 NULL，循环便迭代。循环中的第一条语句增大变量 count 的值。第 35 行的语句计算串 mainStr 中最后一处匹配的索引，它是通过将指针 p 的地址减去变量 mainStr 中第一个字符的地址而获得的(记住，表达式 &mainStr[0] 与简单表达式 mainStr 等价)。语句把结果赋值给变量 index。第 36 行的输出语句显示变量 index 值。

第 36 行的语句寻找串 subStr 在 mainStr 中出现的下一个位置。注意，语句调用 strstr 并提供指针 p 作为第一个变元，还使用前增量运算符使指针 p 存储了下一个字符的地址。这个操作可以保证调用函数找出下一个匹配位置，如果有的话，不至于停留在上一个匹配位置处。whie 循环外的 if 语句检测试变量 count 的值。如果 count 为 0，程序执行第 40 行的输出语句通知用户没有找到与查找串匹配的子串。

第 42 行的输出语句提示程序正在主串中查找指定的字符。查找 findChar 字符的过程与查找串 subStr 的过程很相似。主要的区别是查找字符时调用了函数 strchr。

小　　结

今天的课程讲述了 C++ 串，并讨论了 STRING.H 头文件所提供的串操作函数。主要内容如下:

- C++ 中的串是以空字符(ASCII 码为 0 的字符)结尾的字符数组。
- 串输入需要用 getline 流输入函数。该函数要求指定输入变量、最大输入字符数和任选的行定界符。
- C++ 支持两种串赋值方法。第一种是在声明一个串时把另一个串赋值给它。第二种方法是在程序的任意地方，用函数 strcpy 对一个串赋值。串函数库还提供了函数 strdup 拷贝串并分配所需空间。
- 函数 strlen 返回串的长度。
- 用函数 strcat 和 strncat 可以连接两个串。函数 strncat 可指定连接的字符数。
- 函数 strcmp，stricmp，strncmp，strnicmp 可以完成各种类型的串比较。函数 strcmp 使用每个可能的字符对两个串进行区分大小写的比较。stricmp 是函数 strcmp 对应的进行不区分大小写比较的版本。函数 strncmp 是函数 strcmp 的变种，它用指定数目的字符进行串比较。函数 strnicmp 是函数 strncmp 对应的进行不区分大小写串比较的版本。
- 函数 strlwr 和 strupr 分别将串中字符转换为小写和大写。
- 函数 strrev 颠倒串中字符的顺序。
- 函数 strchr，strrchr，strspn，strcspn 和 strpbrk 允许在串中查找字符和简单的字符模板。
- 函数 strstr 在一个串中查找另一个串。函数 strtok 利用一组指定的定界符把串分解成一些较短的子串。

问 与 答

问: 程序的语句中可以用串文字初始化指针吗?

答: 可以,编译器把串文字中的字符存入内存,并把首地址赋值给指针。举例如下:

```
char * p= "I am a small String";
```

另外,还可以用指针覆盖串常量中的字符。但是,要记住用指针 p 访问的字符串应具有固定的字符数。

问: 语句中可以声明指向文字串的常量指针吗?

答: 可以,这种声明与上面所讲过的那种类似。不过,因为语句声明的是常量指针,所以不能覆盖其包含的初始串中的字符。举例如下:

```
const char * p = "Version 1.0";
```

用 const char 指针保存固定信息和标题。

问: 可以声明指向一组文字串的指针数组吗?

答: 可以,这是用指针数组访问成组的消息、标题或其他固定串的方法。举例如下:

```
char* mainMenu[] = { "File", "Edit", "Search", "View",
                     "Debug", "Options", "Windows", "Help"};
```

这样,元素 p[0] 访问第一个串,P[1] 访问第二个串,依次类推。

问: 如何用 strcmp 从指定的字符序号开始比较两个串?

答: 简单的方法是在函数 strcmp 的变元中加上偏移量。举例如下:

```
char s1[41] = "Visual C+ + ";
char s2[41] = "Visual Basic";
int offset = 7;
int i;
i = strcmp(str1 + offset, str2 + offset);
```

问: 如何用 strncmp 从指定的字符序号开始比较两个串中指定数目的字符?

答: 只需在函数 strncmp 的变元中加上偏移量。举例如下:

```
char s1[41] = "Visual C+ + ";
char s2[41] = "Visual Basic";
int offset = 7;
int num = 3;
int i;
i = strncmp(str1 + offset, str2 + offset, num);
```

作 业

作业部分所提供的测验题有助于巩固理解所学内容,通过练习可以积累运用所学知识的经验。要尽量弄懂测验和练习的答案再继续下一天的课程。附录 B 的“答案”中提供了解答。

测　验

1. 以下程序错在哪里?

```
# include < iostream. h>
# include < string. h>
const int MAX =  10;
main()
{
  char s1[MAX+ 1];
  char s2[] =  "12345678901234567890123456789 0";
  strcpy(s1, s2);
  cout < <  "String 1 is "< <  s1
       < <  "\nString 2 is "< <  s2;
  return 0;
}
```

2. 如何用 strncpy 代替 strcpy 修改上一个问题中的程序?
3. 下述语句中, 变量 i 被赋何值?

```
char s1[] =  "Visual C+ + ";
char s2[] =  "Visual Basic";
int i;
i =  strcmp(s1, s2);
```

4. 下述语句中, 变量 i 被赋何值?

```
char s1[] =  "Visual C+ + ";
char s2[] =  "Visual Basic";
int offset =  strlen( "Visual ");
int i;
i =  strcmp(s1 +  offset, s2 +  offset);
```

5. 下述函数如果串中不包含小写字母则返回 1; 否则返回 0, 对吗?

```
int hasNoLowerCase( const char *  s)
{
char s2[strlen(s)+ 1];
strcpy(s2, s);
strupr(s2);
return ((strcmp(s1, s2) = =  0) ? 1 : 0);
}
```

练　习

1. 自己编写一个 strlen 函数。使用 while 循环和字符计数变量以求得函数结果。
2. 编写另一个版本的 strlen 函数。这次用 while 循环和局部指针来求函数结果。
3. 编写程序 STRING5. CPP, 用函数 strtok 将串 "2* (X+ Y)/(X+ Z) - (X+ 10)/(Y- 5)" 分解为三组标记, 所用标记定界串分别为: "+ - * / ()", "()" 和 "+ - * /"。

第10天

高级函数参数

函数是从概念上对 C++ 语言进行扩展,以适合于用户特定的应用程序需要的基本构件。C++ 的上一代语言,即 C 语言,比 C++ 更面向函数。这种差异是由于对类、继承和其他面向对象程序特性的支持(有关后者的更详细内容将在明天的课程中介绍)。但无论怎样,在 C++ 中,函数依旧扮演着重要角色。今天这一课将学习 C++ 函数的高级应用:

- 传递数组变元
- 传递字符串变元
- 由数值传递结构
- 由引用传递结构
- 由指针传递结构
- 递归函数
- 传递指向动态结构的指针
- 指向函数的指针

10.1 传递数组变元

当编写一个传递数组参数的 C++ 函数时,可以声明一个指向数组基本类型的指针。

语 法

指向数组的指针参数

声明带有指向数组的指针参数的函数原型的一般语法为:

```
returnType function(basicType* , < other parameter types> );
```

定义这类函数的一般语法为:

```
returnType function(basicType * arrParam, < other parameters> )
```

示 例:

```
// prototypes
void ShellSort(unsigned * doubleArray, unsigned arraySize);
void QuickSort(unsigned * intArray, unsigned arraySize);
```

第 7 天讲述了 C++ 允许用一对空的方括号声明开数组参数。这种声明等同于使用指针参数。C++ 程序员较少使用开数组的形式,更多的是用指针形式,虽然用方括号可以更清晰地显示参数的意义。

要用 const 参数禁止宿主函数对变元进行修改。

不要忘记包含一个指定要操作的数组元素的数目的参数，当数组型的变元只是部分填充了有意义的数据时。

我们来看一个例子。清单 10.1 中列出了程序 ADVFUN1.CPP 的源代码。它是通过对程序 ARRAY5.CPP(在第 7 天的程序 7.5 中)进行少量修改而建立的。程序完成下列任务:

- 提示用户输入数据点的数目;
- 提示用户输入数组的整数值;
- 显示无序数组的元素;
- 显示排序后的数组元素。

键　入　　清单 10.1　程序 ADVFUN1.CPP 的源代码

```
1: // C++ program that sorts arrays using the Comb sort method
2:
3: #include <iostream.h>
4:
5: const int MAX = 10;
6: const int TRUE = 1;
7: const int FALSE = 0;
8:
9: int obtainNumData()
10: {
11:    int m;
12:    do { // obtain number of data points
13:       cout << "Enter number of data points [2 to "
14:            << MAX << "]: ";
15:       cin >> m;
16:       cout << "\n";
17:    } while (m < 2 ©©¦m > MAX);
18:    return m;
19: }
20:
21: void inputArray(int * intArr, int n)
22: {
23:    // prompt user for data
24:    for (int i = 0; i < n; i++) {
25:       cout << "arr[" << i << "]: ";
26:       cin >> * (intArr + i);
27:    }
28: }
29:
30: void showArray(const int * intArr, int n)
31: {
```

```
32:    for (int i = 0; i < n; i++) {
33:       cout.width(5);
34:       cout << *(intArr + i) << "";
35:    }
36:    cout << "\n";
37: }
38:
39: void sortArray(int *intArr, int n)
40: {
41:    int offset, temp, inOrder;
42:
43:    offset = n;
44:    do {
45:       offset = (8*offset) / 11;
46:       offset = (offset == 0) ? 1 : offset;
47:       inOrder = TRUE;
48:       for (int i = 0, j = offset; i < (n - offset); i++, j++) {
49:          if (intArr[i] > intArr[j]) {
50:             inOrder = FALSE;
51:             temp = intArr[i];
52:             intArr[i] = intArr[j];
53:             intArr[j] = temp;
54:          }
55:       }
56:    } while (!(offset = 1 && inOrder == TRUE));
57: }
58
59: main()
60: {
61:    int arr[MAX];
62:    int n;
63:
64:    n = obtainNumData();
65:    inputArray(arr, n);
66:    cout << "Unordered array is:\n";
67:    showArray(arr, n);
68:    sortArray(arr, n);
69:    cout << "\nSorted array is:\n";
70:    showArray(arr, n);
71:    return 0;
72: }
```

以下是清单 10.1 中程序的一个运行范例:

输　出

```
Enter number of data points [2 to 10] : 5

arr[0] : 55
arr[1] : 22
arr[2] : 78
arr[3] : 35
arr[4] : 45
```

```
Unordered array is:
   55      22      78      35      45

  Sorted array is:
   22      35      45      55      78
```

分　析

清单 10.1 中的程序与第 7 天的清单 7.5 中的程序基本上相同。新程序中，函数 inputArray，showArray 和 sortArray 中所用的参数稍微有些不同。这些函数的第一个参数是一个指向 int 型的指针。函数 showArray 给指针类型加了前缀 const。这个声明告诉编译器，函数 showArray 不能修改参数 intArray 所代表的变元的值。

10.2　使用字符串变元

因为 C++ 把字符串作为字符数组处理，所以传递数组参数的规则也同样适用于字符串。下一个程序包含了字符串操作函数。清单 10.2 中列出了程序 ADVFUN2.CPP 的源代码。程序提示用户输入一个字符串，然后显示键入的字符个数(输入字符串的长度)和输入字符串的大写版本。

键　入　　清单 10.2　程序 ADVFUN2.CPP 的源代码

```
 1: /*
 2:    C++ program that declares functions with string parameters
 3: */
 4:
 5: #include <iostream.h>
 6:
 7: const unsigned MAX = 40;
 8:
 9: char* upperCase(char* str)
10: {
11:     int ascii_shift = 'A' - 'a';
12:     char* p = str;
13:
14:     // loop to convert each character to uppercase
15:     while (*p != '\0') {
16:         if ((*p >= 'a' && *p <= 'z'))
17:             *p += ascii_shift;
18:         p++;
19:     }
20:     return str;
21: }
22:
23: int strlen(char* str)
24: {
```

```
25:    char * p = str;
26:    while (* p+ + ! = '0');
27:    return p -str;
28:
29:
30: main()
31: {
32:        char aString[Max + 1];
33:
34:        cout < < "Enter a string:";
35:        cin.geline(aString, MAX);
36:        cout < < "Your string has " < < strlen(aString)
37:             < < "characters\n";
38:        // concatenate bigStr to aString
39:        upperCase(aString);
40:        cout < < "The uppercase version of your input is: "
41:             < < aString;
42:        return 0;
43: }
```

以下是清单 10.2 中程序的一个运行范例:

输　出

```
Enter a string : Visual C+ + 1.0
Your string has 14 characters
The uppercase version of your input is : VISUAL C+ + 1.0
```

分　析

清单 10.2 中的程序声明了自己的字符串操作函数: upperCase **和** strlen**。函数** upperCase **带一个参数,即指向** char **类型的指针** str**。此参数传递字符数组的地址。函数把由指针串访问的字符转换为大写并返回指向串的指针。函数声明了局部变量** ascii- shift **和局部字符指针** p**,并用字母** A **与** a **的** ASCII **值的差初始化变量** ascii- shift**。因此,** ascii-shift **变量包含了将小写字母转换为大写字母时所需要的** ASCII **码的差值。函数还用参数** str **的地址初始化局部指针** p**。**

函数 upperCase **在第** 15 **行中用** while **循环对串变元中的字符进行转换。** while **子句判定指针** p **是否未指向空终结符,第** 16 **行的** if **语句判定指针** p **指向的字符是否是小写字母。如果条件为真,函数执行第** 17 **行中的语句,将指针** p **指向的当前字符加上变量** ascii-shift **的值。这一操作把小写字母转换为大写字母。第** 18 **行的语句增加指针** p **的地址指向下一个字符。函数返回指针** str **的地址。**

函数 strlen **返回字符指针参数** str **所指向字符串中的字符数。此函数声明了局部字符指针** p **并将其初始化为参数** str **的地址,函数利用循环语句为空的** while **循环定位由指针** p **所指向的字符串中空终结符的位置。**return **语句通过计算指针** p **和** str **的差产生结果值,并先利用前减量运算符调整指针** p **的地址。**

函数 main 声明了字符串变量 aString。第 34 行的输出语句提示用户输入一个字符串。第 35 行中的语句调用流输入函数 geline 获得输入并存入变量 aString。第 36 行的输出语句显示键入的字符数,该语句调用函数 strlen 并传递参数 aString。第 39 行调用函数 upperCase 并也传递参数 aString。第 40 行的输出语句显示已存于变量 aString 中的输入字符串的大写版本。

10.3　使用结构变元

C++ 允许用值或者引用传递结构。这一节介绍用值传递结构,下一节将学习怎样通过引用传递结构。结构的类型出现在函数原型中并且有与预定义类型中格式相近的头。

清单 10.3 中列出了 ADVFUN3.CPP 的源代码。程序完成下列任务:

- 提示用户输入第一个点的 X 和 Y 坐标;
- 提示用户输入第二个点的 X 和 Y 坐标;
- 计算输入的两个点的中点坐标;
- 显示中点的坐标。

键　入　　清单 10.3　程序 ADVFUN3.CPP 的源代码

```
1: // C++ program which uses a function that passes
2: // a structure by value
3:
4: #include <iostream.h>
5:
6: struct point {
7:    double x;
8:    double y;
9: };
10:
11: // declare the prototype of function getMedian
12: point getMedian(point, point);
13:
14: main()
15: {
16:    point pt1;
17:    point pt2;
18:    point median;
19:
20:    cout << "Enter the X and Y coordinates for point # 1: ";
21:    cin >> pt1.x >> pt1.y;
22:    cout << "Enter the X and Y coordinates for point # 2: ";
23:    cin >> pt2.x >> pt2.y;
24:    // get the coordinates for the median point
25:    median = getMedian(pt1, pt2);
26:    // get the median point
27:    cout << "Mid point is (" << median.x
28:         << ", " << median.y << ")\n";
```

```
29:    return 0;
30: }
31:
32: point getMedian(point p1, point p2)
33: {
34:    point result;
35:    result.x = (p1.x + p2.x) / 2;
36:    result.y = (p1.y + p2.y) / 2;
37:    return result;
38: };
```

以下是清单 10.3 中程序的一个运行范例:

输　出

```
Enter the X and Y coordinates for point # 1: 1 1
Enter the X and Y coordinates for point # 2: 5 5
Mid point is (3,3)
```

分　析

清单 10.3 中的程序声明了建立二维点模型的结构 point。**这个结构有两个** double **类型的成员:** x **和** y。**第 12 行声明了函数** geMedian **的原型,该函数使用了两个用值传递的** point **类型参数。**

函数 main **在第 16 到 18 行分别声明了** point **型变量** pt1, pt2 **和** median。**第 20 行的输出语句提示用户输入第一个点和** x **和** y **坐标。第 21 行的语句获得输入并把坐标存入成员** pt1.x **和** pt1.y。**第 22 和 23 行对第 2 个点重复同样的提示和并进行输入处理。第 23 行的输入语句把值存入第 2 个点的成员** pt2.x **和** pt2.y **中。第 25 行的语句调用函数** getMedian **并传递了参数** pt1 **和** pt2。**函数接收到参数** pt1 **和** pt2 **的副本,并把** point **类型的函数结果赋值给变量** median。**第 27 和 28 行的输出语句显示中点的** X **和** Y **坐标(通过显示变量** median **的成员** x **和** y)。

函数 getMedian **声明了两个** point **类型的参数:** p1 **和** p2。**另外,该函数还声明了局部** point **型变量** result。**第 35 行的语句把成员** p1.x **和** p2.x **的平均值赋值给** result.x。**第 37 行的语句把成员** p1.y **和** p2.y **的平均值赋值给** result.y。**注意,这些语句使用了点运算符存取结构** p1, p2 **和** result **的成员** x **和** y。**当函数传递结构的副本或引用时要用到这一语法。返回语句返回局部结构** result **的值。**

10.4　由引用传递变元

用 C+ + **可以编写由引用传递变元的函数。这种参数可以使程序员越过函数的作用域而修改变元的值。**C+ + **提供了两种使用这种参数的方法:使用指针和使用正规引用参数。下面各小节介绍由引用传递各种数据类型的函数。**

10.5　由引用传递结构

可以用指针或正规引用向函数传递结构。多数 C+ + 程序员认为这两种方法中的任一种都比通过副本传递结构参数要高效得多,因为这样可节省复制结构类型参量的开销。

> **注　意**　通过引用传递结构时可以对结构参数使用点运算符(与传递值一样)。其优点在于引用参数不产生原始变元的副本,因而这种方法更快,也更节省内存资源。其缺点在于因为引用参数成为其变元的别名,所以函数中任何对引用参数的改变都会影响到实参。防止这种修改的一种方法是使用 const 引用参数,告诉编译器函数不能对引用参数赋新值。

> **要**通过正规引用或指针传递结构,对于下面两种情况:宿主函数不修改变元以及函数通过这些结构返回结果。
>
> **不要**传递结构的值,除非需要向宿主函数提供它要修改的数据的副本。

我们来看一个例子。清单 10.4 中列出了程序 ADVFUN4.CPP 的源代码。程序所完成的工作与上一个程序 ADVFUN3.CPP 相同。新版本仅在实现方法上有差别。

键　入　　**清单 10.4　程序 ADVFUN4.CPP 的源代码**

```
 1: // C+ + program which uses a function that passes
 2: // a structure by reference
 3:
 4: #  include < iostream.h>
 5:
 6: struct point {
 7:    double x;
 8:    double y;
 9: }
10:
11: // declare the prototype of function getMedian
12: point getMedian(const point&, const point&);
13:
14: main()
15: {
16:    point pt1;
17:    point pt2;
18:    point median;
19:
20:    cout < <  "Enter the X and Y coordinates for point #  1: ";
21:    cin > >  pt1.x > >  pt1.y;
22:    cout < <  "Enter the X and Y coordinates for point #  2: ";
```

```
23:    cin >> pt2.x >> pt2.y;
24:    // get the coordinates for the median point
25:    meian = getMedian(pt1, pt2);
26:    // get the median point
27:    cout << "Mid point is (" << meidan.x
28:         << ", " << meidan.y << ")\n";
29:    return 0;
30: }
31:
32: point getMedian(const point& p1, const point& p2)
33: {
34:    point result;
35:    result.x = (p1.x + p2.x) / 2;
36:    result.y = (p1.y + p2.y) / 2;
37:    return result;
38: };
```

以下是清单 10.4 中程序的一个运行范例:

输　出

```
Enter the X and Y coordiantes for point # 1: 1 1
Enter the X and Y coordinates for point # 2: 9 9
Mid point is (5, 5)
```

分　析

清单 10.4 程序与清单 10.3 很相似。新版本程序在函数 getMedian 中用了引用参数,因而函数原型和函数声明中将 & 字符置于结构类型 point 后面。使用引用参数对函数 getMedian 的调用看起来很像清单 10.3 中的调用。同样,函数 getMedian 使用的工具也与清单 10.3 中的相似——在结构 point 中都使用了点运算符存取成员 x 和 y。

10.6　用指针传递结构

使用指针是传递结构的另一种有效方法。像引用参数类型那样,使用 const 声明可以防止修改由指针参数所指向的结构变量。

下一个例子是程序 ADVFUN3.CPP 使用指针参数的版本。清单 10.5 中列出了新版本,ADVFUN5.CPP 的源代码。

键　入　　清单 10.5　程序 ADVFUN5.CPP 的源代码

```
1: // C++ program which uses a function that passes
2: // a structure by pointer
3:
4: # include <iostream.h>
```

```
 5:
 6: struct point {
 7:    double x;
 8:    double y;
 9: }
10:
11: // declare the prototype of function getMedian
12: point getMedian(const point* , const point* );
13:
14: main()
15: {
16:    point pt1;
17:    point pt2;
18:    point median;
19:
20:    cout << "Enter the X and Y coordinates for point # 1: ";
21:    cin >> pt1.x >> pt1.y;
22:    cout << "Enter the X and Y coordinates for point # 2: ";
23:    cin >> pt2.x >> pt2.y;
24:    // get the coordinates for the median point
25:    median = getMedian(&pt1, &pt2);
26:    // get the median point
27:    cout << "Mid point is (" << meidan.x
28:         << ", " << median.y << ")\n";
29:    return 0;
30: }
31:
32: point getMedian(const point* p1, const point* p2)
33: {
34:    point result;
35:    result.x = (p1-> x + p2-> x) / 2;
36:    result.y = (p1-> y + p2-> y) / 2;
37:    return result;
38: };
```

以下是清单 10.5 中程序的一个运行范例:

输　出

```
Enter the X and Y coordiantes for point # 1: 2 2
Enter the X and Y coordinates for point # 2: 8 8
Mid point is (5, 5)
```

分　析

清单 10.5 中的程序在函数 getMedian 中使用了指针参数。函数原型及其实现中所使用的参数都是 const point * 类型的。第 25 行的语句调用函数 getMedian, 并使用取地址运算符 & 传递 pt1 和 pt2 的地址。函数 getMedian 用 -> 运算符存取指针参数 p1 和 p2 的成员 x 和 y。

10.7　递 归 函 数

有许多问题可以被分解为简单的且类似的小问题。这种问题可以用递归的方法解决。

新　词

递归函数(recursive function)
递归函数是通过调用自身获得结果或完成一项任务的函数。这种递归调用必须有所限制以免耗尽计算机的内存资源。因此，每个递归函数都必须要检测判断递归终止条件。

新　词

阶乘函数(factorial function)
递归函数的一个常见的例子就是阶乘函数。数 N 的阶乘是从 1 到 N 的所有整数的乘积，阶乘函数的数学符号用惊叹号！表示。

阶乘的数学公式是：

```
N! = 1 * 2 * 3 * … * (N- 2) * (N- 1) * N
```

这一等式的递归形式为：

```
N! = N * (N- 1)!
(N- 1)! = (N- 1) * (N- 2)!
(N- 2)! = (N- 2) * (N- 3)!
…
2! = 2 * 1!
1! = 1
```

递归通过循环调用获得结果。大多数的递归方法都可转化为非递归方法。在一些情况下，递归解法要比相应的非递归解法优美得多。阶乘函数便是一个可以使用递归或非递归直接循环的方法解决的数学函数的例子。

要在递归函数中包含一个决策结构语句以便结束递归。

不要使用递归——如果递归法比起非递归法没有显著的优越性的话。

我们来看一个实现递归阶乘函数的例子。清单 10.6 中列出了程序 ADVFUN6.CPP 的源代码。程序提示用户输入两个正整数——第 1 个必须大于或等于第 2 个整数。程序显示两个整数的组合和排列，求组合的等式如下：

```
mCn = m! / ((m- n)! * n!)
```

求排列数的等式如下：

```
mPn = m! / (m- n)!
```

键　入　　**清单 10.6　程序 ADVFUN6.CPP 的源代码**

```
// C++ program which uses a resursive function

#include <iostream.h>

const int MIN = 4;
const int MAX = 30;

double factorial(int i)
{
   if (i > 1)
      return double(i) * factorial(i - 1);
   else
      return 1;
}

double permutation(int m, int n)
{
   return factorial(m) / factorial(m -n);
}

double combination(int m, int n)
{
   return permutation(m, n) / factorial(n);
}

main()
{
   int m, n;

   do {
      cout << "Enter an integer between "
           << MIN << "and "<< MAX << ": ";
      cin >> m;
   } while (m < MIN ©©¦m > MAX);

   do {
      cout << "Enter an integer between "
           << MIN << "and "<< m << ": ";
      cin >> n;
   } while (n < MIN ©©¦n > m);

   cout << "Permutations("<< m << ", "<< n
        << ") = "<< permutation(m, n) << "\n";
   cout << "Combinations("<< m << ", "<< n
        << ") = "<< combination(m, n) << "\n";

    return 0;
}
```

以下是清单 10.6 中程序的一个运行范例:

输　出

```
Enter an integer between 4 and 30 : 10
Enter an integer between 4 and 10 : 5
Permutations(10, 5) = 30240
Combinations(10, 5) = 252
```

分　析

清单 10.6 中的程序声明了递归函数 factorial 以及函数 permutation，combination 和 main。函数还声明了全局常量 MIN 和 MAX,它们限制用户键入的第 1 个数的范围。

函数 factorial 有一个 int 型的参数 i。函数返回的是 double 型的值。第 10 行的 if 语句将参数 i 的值与 1 比较,决定是否进行第 11 行的递归调用或者转第 13 行返回数值 1。第 11 行的递归调用以变元 i- 1 调用函数 factorial。这样,递归调用时提供的是更小的(或者是更简单的)变元。

函数 permutation 有两个 int 型的参数:m 和 n。函数两次调用递归函数 factorial:一次是以 m 为变元,一次是以 m- n 为变元。函数 permutation 返回函数 factorial 两次调用的比值。

函数 combination 也有两个 int 型的参数:m 和 n,它调用函数 permutation 并以 m 和 n 为变元。该函数还调用函数 factorial 并传递变元 n。函数 combination 返回函数 permutation 和 factorial 所返回值的比值。

函数 main 中声明了 int 型的变量 m 和 n。函数用两个 do-while 循环提示用户输入整型变量。第 1 个循环中的输出语句提示用户输入一个范围在 MIN 和 MAX 之间的整数。第 33 行的语句保存输入变量 m 的值。do-while 循环的 while 子句对输入进行验证,判断是否输入值小于 MIN 或大于 MAX。如果条件为真,循环再次迭代。

第 2 个 do-while 循环中的输出语句提示用户输入一个范围在 MIN 和 m 之间的整数。第 39 行的语句把输入存储到变量 n 中。while 子句检查输入合法性,判定输入是否小于 MIN 或大于 m。如果条件为真,循环再次迭代。

第 42 和 43 行的输出语句显示变量 m 和 n 中的排列值,该语句调用函数 permutation 并传递变元 m 和 n。

10.8　传递指向动态结构的指针

实现二叉树至少需要的功能有插入、查找、删除和遍历。所有这些功能都要通过根指针访问二叉树。有趣的是,诸如树的插入和删除这样的操作可能影响到根本身。在这种情况下根结点的地址改变了。因此,需要传递指向根结点的指针的引用,使用指向指针的引用可以保证对树根结点地址的修改。

新　词

二叉树(binary tree)　　**结点**(node)　　**根**(root)　　**域**(field)

二叉树是常用的动态数据结构。这样的结构可以在没有数据项数目信息的情况下建立有序数据集合。二叉树中的每一个结点都是它下面的子树的根。终结点是空子树的根。二叉树有一个特殊的结点,它是所有其他结点的根。每个结点有一个域(作为排序关键字)、任选的附加数据(称为非关键数据)以及与其他树结点建立连接的两个指针。可以用动态内存分配为每个结点分配空间,并在不同的结点间动态地建立连接。为了更多的掌握二叉树的结构,可以查阅有关数据结构的书。

要使用指针的引用声明处理指向数据结构的指针的参数。这一声明可保证越过函数的作用域修改参数的地址。

不要认为函数对非引用指针参数的地址进行的修改会修改影响到相应变元的地址。

我们来看在二叉树中插入和显示动态数据的例子。清单 10.7 中的程序列出了程序 ADVFUN7.CPP 的源代码。程序自己提供了数据(一组名字),在二叉树中插入数据并按升序排列二叉树中的数据。

键　入　　**清单 10.7　程序 ADVFUN7.CPP 的源代码**

```
1: // C++ program which passes parameter to dynamic data
2:
3: #include <iostream.h>
4: #include <string.h>
5:
6: const unsigned MAX = 30;
7:
8: typedef struct node* nodeptr;
9:
10: struct node {
11:     char value[MAX+1];
12:     nodeptr left;
13:     nodeptr right;
14: };
15:
16: void insert(nodeptr& root, const char* item)
17: // Recursively insert element in binary tree
18: {
19:   if (!root)  {
20:     root = new node;
21:     strncpy(root->value, item, MAX);
22:     root->left = NULL;
23:     root->right = NULL;
24:   }
25:   else {
26:     if (strcmp(item, root->value) < 0)
```

```
27:          insert(root -> left, item);
28:       else
29:          insert(root -> right, item);
30:    }
31: }
32:
33: void showTree(noedptr& root)
34: {
35:    if (! root)
36:       return;
37:
28:    showTree(root -> left);
39:    cout < < root -> value < < "n";
40:    showTree(root -> right);
41: }
42:
43: main( )
44: {
45:    char * names[] = { "Virginia", "California", "Maine", "Michigan",
46:                       "New York", "Florida", "Ohio", "Illinois",
47:                       "Alaska", "Arizona", "Oregon", "Vermont",
48:                       "Maryland", "Delaware", "NULL "};
49:    nodeptr treeRoot = NULL;
50:    int i = 0;
51:
52:    // insert the names in the binary tree
53:    while (strcmp(name[i], "NULL ") ! = 0)
54:       insert(treeRoot, names[i+ + ]);
55:
56:    showTree(treeRoot);
57:    return 0;
58: }
```

以下是清单 10. 7 中程序的一个运行范例:

输　出

```
Alaska
Arizona
California
Delaware
Florida
Illinois
Maine
Maryland
Michigna
New York
Ohio
Oregon
Vermont
Virginia
```

分　析

清单 10.7 中的程序声明了全程常量 MAX, 用以指定二叉树中每一结点中的最大字符个数。第 8 行的声明定义了基于结点结构的指针类型 nodeptr。程序在第 10 到 14 行定义了结构 node, 其中包含成员 value(存储字符串)、指向左结点的指针 left 和指向右结点的指针 right。两个指针都是 nodeptr 类型的。

程序声明了递归函数 insert 以在二叉树中插入字符串。该函数有两个参数: root 和 item。参数 root 是一个 nodeptr 类型指针的引用, 记录了二叉树各结点的路径并在需要时修改它们的地址。这种修改发生在二叉树插入数据项时。

第 19 行的 if 语句判定参数 root 是否为空。如果条件为真, 函数执行第 20 到 23 行的语句。第 20 行的语句用运算符 new 分配一个新结点。第 21 行中的语句使用函数 strncpy 从参数 item 中拷贝 MAX 个字符到成员 value 中。第 22 和 23 行的语句对新创建结点的左右结点指针赋值为 NULL。第 20 到 24 行的语句不仅影响到树根, 也修改了指向各结点的指针。if 语句中的条件协助结束递归调用。

第 25 行中的 else 子句处理参数 root 不为空的情况。第 26 行的 if 语句判定指针 item 所指向的串是否小于当前树结点 value 成员中的字符串。如果条件为真, 函数产生递归调用并传递变元 root -> left 和 item。这一调用在左子树中插入了新字符串, 它的树根是当前结点; 否则, 函数产生递归调用并传递变元 root-> right 和 item。这一调用在右子树中插入了新字符串, 并以当前结点为根。

递归函数 showTree 按顺序遍历以引用参数 root 为根的树及其子树的结点。如果参数 root 的当前值为 NULL, 函数立即退出, 此条件表明参数 root 的变元是终结点, 因此结束递归调用。如果参数 root 的变元不为 NULL, 函数进行递归调用并传递变元 root -> left。此调用允许函数访问以当前结点为根的左子树, 一旦访问了当前结点的左子树, 函数便显示存储在当前结点的 value 成员中的字符串。然后函数进行另一递归调用, 这次传递变元 root -> right。这一调用可以使函数访问以当前结点为根的右子树, 访问了当前结点的右子树后, 函数退出。

函数 main 中声明了指向内部数据的数组, 见第 45 到 48 行。该函数还声明了变量 treeRoot 作为二叉树的根并被初始化为 NULL。函数还声明了 int 型变量 i 并初始化为 0。

函数 main 在第 53 行中用 while 循环插入数据表中的州名字。循环一直迭代到当前名字与串 "NULL "匹配, 这一字符串用于标记表尾。如果想修改程序以增添更多的州名, 一定要保证表中的最后一项为 "NULL "串。第 54 行的语句调用函数 insert 将元素 name [i] 插入以指针 treeRoot 为根的二叉树。

第 56 行的语句调用函数 showTree 并以 treeRoot 为变元, 此递规调用将以升序显示州名。

10.9　指向函数的指针

程序的编译过程把变量名字转换为存储就检索数据的内存地址。指向地址的指针也

可以访问这些地址。这一转换过程对变量和函数都是正确的。编译器将函数名转换为可执行代码的地址。C++ 通过使用指向函数的指针扩展了变量处理策略。

语　法

指向函数的指针

声明指向函数的指针的一般语法:

```
returnType (* functionPointer)(< list of parameters> );
```

这一形式告诉编译器 functionPointer 是一个具有 returnType 返回类型和< list of parameters> 参数表的函数指针。

示　例:

```
double (* fx)(int n);
void (* sort) (int* intArray, unsigned n);
unsigned (* search)(int searchKey, int* intArray, unsigned n);
```

第 1 个标识符 fx 指向一个返回 double 类型并有一个 int 类型参数的函数。第 2 个标识符 sort 是一个函数指针,返回 void 类型并有两个参数:指向 int 类型的指针和 unsigned 类型参数。第 3 个标识符是一个返回 unsigned 的函数指针,它有 3 个参数:int 类型的参数、指向 int 的指针和 unsigned 类型参数。

语　法

函数指针数组

C++ 允许声明函数指针数组,一般语法为:

```
returnType (* functionPointer[arraySize]) (< list of parameters> );
```

示　例:

```
double (* fx[3])(int n);
void (* sort[MAX_SORT])(int* intArray, unsigned n);
unsigned (* search[MAX_SEARCH] (int searchKey,
                                int* intArray, unsigned n);
```

第 1 个标识符 fx 指向一个函数数组。每一个成员有一个 int 类型参数并返回 double 型数据。第 2 个标识符 sort 是一个函数数组指针,每个成员返回 void 类型并有两个参数:int 类型指针和 unsigned 类型参数。第 3 个标识符 search 是一个函数数组指针,每一成员返回 unsigned 型结果并有 3 个参数:int 类型参数,指向 int 类型的指针和 unsigned 类型参数。

像所有指针一样,在使用函数指针之前也要做初始化。步骤很简单,只需把一个函数的名字赋值给函数指针。

语　法

初始化函数指针

初始化函数指针的一般语法为:

```
functionPointer = aFunction;
```

被赋值的函数必须有和该函数指针相同的返回值和参数表;否则,编译器产生错误。

示　例:

```
void (* sort)(int* intArray, unsigned n);
sort = qsort;
```

语　法

将函数赋值给元素

将函数赋值给函数数组指针中的一个元素的一般语法为:

```
functionPointer[index] = aFunction;
```

一旦把函数名赋值给函数指针,便可以用指针调用相应函数。现在便清楚了为什么函数指针要与被访问的函数具有相同的返回值和参数表。

示　例:

```
void (* sort[2])(int* intArray, unsigned n);
sort[0] = shellSort;
sort[1] = CombSort;
```

语　法

函数指针表达式

使用函数指针的表达式的一般语法为:

```
(* functionPointer)(< argument list> );
(* functionPointer[index])(< argument list> );
```

示　例:

```
(* sort)(&intArray, n);
(* sort[0])(&intArray, n);
```

我们来看一个例子,清单 10.8 中列出了程序 ADVFUN8.CPP 的源代码。程序对以下两个观察变量进行线性拟合:独立变量 X 和独立变量 Y。与两个变量相关的模型是:

```
f(Y) = intercept + slope * g(X)
```

函数 f(Y)变换变量 Y 中的数据,函数 g(X)变换变量 X 中的数据。函数 f(Y)和 g(X)可以是线性、对数、指数、平方根、平方或其他数学函数。当 f(Y) = Y 且 g(X) = X 时,模型便成为线性拟合模型:

```
Y = intercept + slope * X
```

线性拟合(一般模型)计算对 f(Y)和 g(X)的最佳 slope 和 intercept 值,拟合也提供了代表当前 g(X)数据变动对 f(Y)的影响程度的相关系数。1 代表完全拟合;0 表示 f(Y)与 g(X)的数据相互独立。

清单 10.8 完成线性拟合并执行下列任务:

- **提示输入数据数目(输入必须在程序所示的限制范围中);**
- **提示输入 X 和 Y 的观察值;**
- **提示选择变量 X 的变换和观察函数(程序显示供用户选择的小的菜单项,包括线性、对数、平方根、平方、和倒数函数);**
- **提示选择变量 Y 的变换和观察函数(程序显示供用户选择的小的菜单项,包括线性、对数、平方根、平方、和倒数函数);**
- **进行拟合计算;**

· 显示线性拟合的 intercept，slope 以及相关系数的值
· 提示选择另一变换函数集(如果选择用另一函数集，程序从第 3 步起继续执行)。

键　入　　清单 10.8　程序 ADVFUN8.CPP 的源代码

```
/*
    C++ program that uses pointers to functions to implement
    a linear regression program that supports temporary
    mathematical transformations.
*/

#include <iostream.h>
#include <math.h>

const unsigned MAX_SIZE = 100;

typedef double vector[MAX_SIZE];

struct regression {
    double Rsqr;
    double slope;
    double intercept;
};

// declare function pointer
double (*fx) (double);
double (*fx)(double);

// declare function prototypes
void initArray(double*, double*, unsigned);
double linear(double);
double sqr(double);
double reciprocal(double);
void calcRegression(double*, double*, unsigned, regression&,
                    double (*fx)(double), double (*fx)(double));
int select_transf(const char*);

main()
{
    char ans;
    unsigned count;
    vector x, y;
    regression stat;
    int trnsfx, trnsfy;

    do {
        cout << "Enter array size [2.."
             << MAX_SIZE << "]: ";
        cin >> count;
    } while (count <= 1 ©©¦count > MAX_SIZE);

```

```
47:         // initialize array
48:         initArray (x, y, count);
49:         // transform data
50:         do {
51:           // set the transformation functions
52:           trnsfx =  select_transf( "X ");
53:           trnsfy =  select_transf( "Y ");
54:           // set function pointer fx
55:           switch (trnsfx) {
56:           case 0:
57:             fx =  linear;
58:             break;
59:           case 1:
60:             fx =  log;
61:             break;
62:           case 2:
63:             fx =  sqrt;
64:             break;
65:           case 3:
66:             fx =  sqr;
67:             break;
68:           case 4:
69:             fx =  reciprocal;
70:             break;
71:           default:
72:             fx =  linear;
73:             break;
74:           }
75:           // set function pointer fy
76:           switch (trnsfy) {
77:           case 0:
78:             fy =  linear;
79:             break;
80:           case 1:
81:             fy =  log;
82:             break;
83:           case 2:
84:             fy =  sqrt;
85:             break;
86:           case 3:
87:             fy =  sqr;
88:             break;
89:           case 4:
90:             fy =  reciprocal;
91:             break;
92:           default:
93:             fy =  linear;
94:             break;
95:           }
96:
97:           /*  call function with functional arguments
98:
99:                                              ↓  ↓   */
100:          calcRegression(x, y, count, stat, fx, fy);
```

```
101:
102:          cout << "\n\n"
103:               << "R-square = " << stat.Rsqr << "\n"
104:               << "Slope = " << stat.slope << "\n"
105:               << "Intercept = " << stat.intercept << "\n\n\n";
106:          cout << "Want to use other transformations? (Y/N) ";
107:          cin >> ans;
108:        } while (ans == 'Y' || ans == 'y');
109:     return 0;
110: }
111:
112: void initArray(double* x, double* y, unsigned count)
113: // read data for array from the keyboard
114: {
115:      for (unsigned i = 0; i < count; i++, x++, y++) {
116:          cout << "X[" << i << "] : ";
117:          cin >> *x;
118:          cout << "Y[" << i << "] : ";
119:          cin >> *y;
120:    }
121: }
122:
123: int select_transf(const char* var_name)
124: // select choice of tranformation
125: {
126:
127:     int choice = -1;
128:     cout << "\n";
129:     cout << "select transformation for variable " << var_name
130:          << "\n"
131:          << "0) No tranformation\n"
132:          << "1) Logarithmic transformation\n"
133:          << "2) Square root transformation\n"
134:          << "3) Square transformation\n"
135:          << "4) Reciprocal transformation\n";
136:     while (choice < 0 || choice > 4) {
137:        cout << "\nSelect choice by number : ";
138:        cin >> choice;
139:     }
140:     return choice;
141: }
142:
143: double linear(double x)
144: { return x;}
145:
146: double sqr(double x)
147: { return x * x;)
148:
149: double reciprocal(double x)
150: { return 1.0 / x;}
151:
152: void calcRegression (double * x,
153:                      double * y,
154:                      unsigned count,
```

```
155:                        regression &stat,
156:                        double (* fx)(double),
157:                        double (* fy)(double))
158:
159: {
160:     double meanx, meany, sdevx, sdevy;
161:     double sum = (double) count, sumx = 0, sumy = 0;
162:     double sumxx = 0, sumyy = 0, sumxy = 0;
163:     double xdata, ydata;
164:
165:     for (unsigned i = 0; i < count; i++) {
166:         xdata = (* fx)(* (x+ i));
167:         ydata = (* fy)(* (y+ i));
168:         sumx += xdata;
169:         sumy += ydata;
170:         sumxx += sqr(xdata);
171:         sumyy += sqr(ydata);
172:         sumxy += xdata * ydata;
173:     }
174:
175:     meanx = sumx / sum;
176:     meany = sumy / sum;
177:     sdevx = sqrt((sumxx - sqr(sumx) / (sum- 1.0));
178:     sdevy = sqrt((sumyy - sqr(sumy) / (sum- 1.0));
179:     stat.slope = (sumxy - meanx * meany * sum) /
180:                         sqr(sdevx) / (sum- 1);
181:     stat.intercept = meany - stat.slope * meanx;
182:     stat.Rsqr = sqr(sdevx / sdevy * stat.slope);
183:
184: }
```

以下是清单 10.8 中程序的一个运行范例:

输　出

```
Enter array size [2..100]: 5
X[0]: 10
Y[0]: 50
X[1]: 25
Y[1]: 78
X[2]: 30
Y[2]: 85
X[3]: 35
Y[3]: 95
X[4]: 100
Y[4]: 212

select transformation for variable X
0) No transformation
1) Logarithmic transformation
2) Square root transformation
3) Square transformation
4) Reciprocal transformation
```

```
Select choice by number : 1

select transformation for variable Y
0) No transformation
1) Logarithmic transformation
2) Square root transformation
3) Square transformation
4) Reciprocal transformation

Select choice by number : 1

R-square = 0.977011
Slope = 0.63039
Intercept = 2.37056

Want to use other transformations? (Y/N) y

select transformation for variable X
0) No transformation
1) Logarithmic transformation
2) Square root transformation
3) Square transformation
4) Reciprocal transformation

Select choice by number : 0

select tranformation for varialbe Y

0) No transformation
1) Logarithmic transformation
2) Square root transformation
3) Square transformation
4) Reciprocal transformation

Select choice by number : 0

R-square = 0.999873
Slope = 1.79897
Intercept = 32.0412

Want to use other transformations? (Y/N) n
```

分　析

清单 10.8 中的程序声明了决定数组最大长度的全程常量 MAX_SIZE。在第 12 行声明了类型 vector。另外，程序还在第 14 到 18 行定义了结构 regression。这一结构用于拟合统计。第 21 和 22 行定义了全程函数指针 fx 和 fy。每一个指针处理一个带有 double 类型变元并返回 double 类型数值的函数。程序用这些指针存储所选择的数学变换函数。

程序还声明了函数 initArray，linear，sqr，reciprocal，calcRegression，select_transf 和 main。函数 initArray 提示用户输入数组 x 和 y 的数据。函数 linear，sqr 和 reciprocal

是提供数据变换的简单函数。这些函数是对数学函数,如 sqrt 和 log 的补充,数学函数的原型在 MATH.H 头文件中。这些函数与函数指针 fx 和 fy 具相同的参数和返回类型。

函数 calcRegression 根据参数 x 和 y 所传递的数组计算拟合统计。函数利用函数指针 fx 和 fy 变换数组 x 和 y 中的数据。第 166 和 167 行的语句分别用指针 fx 和 fy 变换数组 x 和 y 的元素。

函数 select_transf 通过简单菜单项进行提示,并通过数值选择变换函数。函数返回所选择的变换代码的值。

函数 main 利用类型 vector 声明了数组 x 和 y。函数还声明了存储拟合统计的结构 stat。函数 main 提示用户输入希望处理的数据点个数,然后调用函数 initArray 获得数组 x 和 y 的数据。接着,两次调用函数 select_trans 选择数组 x 和 y 的转换函数。第 55 行的 switch 语句检测变量 trnsfx 的值,其中包含了数组 x 中变换值的索引。各 case 标号把相应的函数赋值到指针 fx。这些函数中的一些,如 log 和 sqrt,其原型在 MATH.H 头文件中。第 76 行的 switch 语句执行相似的任务,把相应的函数赋值给指针 fy。

函数 main 调用 calcRegression 并传送变元 x, y, count, stat, 函数指针 fx 和函数指针 fy。第 102 到 105 行的输出语句显示当前变换函数的拟合统计。第 106 行中的语句询问是否希望选择另一组变换函数。第 107 行的语句把输入存储到变量 ans 中。第 108 行的 while 子句判定是否重复选择变换函数与计算拟合统计的过程。

小　　结

今天这一课讲述了简单的 C++ 功能,主要内容如下:

- 可以用指向基本类型的指针传递数组作为函数的变元。C++ 允许使用显式指针类型或空方括号声明数组参数。用这类参数可以编写适合于不同长度数组的通用函数。此外,这些指针通过地址存取数组元素,而不用复制整个数组。
- 因为 C++ 把字符串作为字符数组,所以传递字符串参数与传递数组参数遵循相同的规则。
- 传递结构作为函数的变元可以将各相关信息封装在 C++ 结构中,从而缩短了参数表长度。C++ 支持用值传递结构。这种参数传送的是宿主函数变元的副本。因此,对结构成员的修改不影响函数作用域外的变元。
- 可以利用指针和正规引用传递参数。正规引用是变元的别名,对参数所作的任何修改都会影响到函数作用域外的变元。可以通过声明常量引用参数来保证函数不会修改引用参数所对应的变元。访问结构引用参数的成员时应使用点运算符。
- 用指针传递结构时宿主函数得到的是结构的地址。指针参数需要用 -> 运算符访问结构的各成员。可以在指针参数前使用 const 前缀来防止函数对指针参数所指结构成员的修改。
- 递归函数是通过调用自身获得结果或完成某项任务。递归调用必须有所限制以免耗尽计算机的内存资源。因此,每一个递归函数都必须检测一个决定递归结束的条件。

· 传递指向动态结构的指针一般要求传递管理这一结构的头或尾指针的引用。今天的课程介绍了如何创建在二叉树中插入数据以及显示数据的函数。
· 函数指针存储函数的地址。这种指针需要有参数表和返回值类型，以便以相同的原型访问函数。函数指针使运行时动态选择调用函数成为可能。

问　与　答

问：与值参相比，如何利用引用参数优化函数设计？
答：引用参数可以修改变元，除非将其声明为 const 参数。这样，函数可以用引用参数作为输入数据通道，也可以作为输出数据通道。
问：如何区分传送数值数组的指针和通过变元回传值的指针？
答：需要根据上下文阅读函数的声明。但是，可以用引用参数声明一个向调用者回传值的参数。
问：用于管理递归函数调用的内存资源是什么？
答：运行系统用推栈保存中间结果，包括调用递归函数时产生的中间结果。像其他内存资源一样，堆栈的空间有限，因此过长序列的递归调用或大量占用内存的变元都会耗尽堆栈空间而导致运行错误。

新　词

堆栈(stack)
堆栈是信息以后进先出(LIFO)顺序进行信息插入和删除的内存单元。

作　　业

作业部分所提供的测验题有助于巩固理解所学内容，通过练习可以积累运用所学知识的经验。要尽量弄懂测验和练习的答案后再继续下一天的课程。附录 B 的“答案”中提供了解答。

测　验

1. 可以用条件运算符编写递归的阶乘函数吗？
2. 下面的递归函数中有何错误？

```
double factorial(int i)
{
switch (i) {
   case 0:
   case 1:
          return 1;
          break;
   case 2:
          return 2;
          break;
   case 3:
```

```
            return 6;
            break;
        case 4:
            return 24;
            break;
        default:
            return double(i) * factorial(i-1);
    }
}
```

3. **把下面的 Fibonacci 递归函数(此函数的数列为 Fib(0) = 0, Fib(1) = 1, Fib(2) = 1, Fib(3) = 2, Fib(4) = 3, 等等) 转换为非递归形式。**

```
double Fibonacci(int n)
{
  if (n == 0)
    return 0;
  else if (n == 1 ©©¦n == 2)
    return 1;
  else
    return Fibonacci(n- 1) + Fibonacci(n- 2);
}
```

4. **对或错? 下面的函数的两种版本是等价的:**

```
struct stringStruct {
        char source[MAX+ 1];
        char uprStr[MAX+ 1];
        char lwrStr[MAX+ ];
        char revStr[MAX+ 1];
};

void convertStr1(const char* str, stringStruct& s)
{
  strncpy(s.source, str, MAX);
  strncpy(s.uprStr, str, MAX);
  strncpy(s.lwrStr, str, MAX);
  strncpy(s.revStr, str, MAX);
  - strlwr(s.lwrStr);
  - strupr(s.uprStr);
  strrev(s.revStr);
}

void convertStr2(const char* str, stringStruct* s)
{
  strncpy(s- > source, str, MAX);
  strncpy(s- > uprStr, str, MAX);
  strncpy(s- > lwrStr, str, MAX);
  strncpy(s- > revStr, str, MAX);
  - strlwr(s- > lwrStr);
  - strupr(s- > uprStr);
  strrev(s- > revStr);
}
```

练　习

根据程序 ADVFUN8. CPP 建立程序 ADVFUN9. CPP, 用函数指针数组 f 替换单独的函数指针 fx 和 fy。另外, 用一个函数指针数组替换函数 calcRegresion 中的两个函数指针参数。

第11天

面向对象的程序设计和C++类

类为C++提供了面向对象的程序设计(OOP)的构件。今天这一课是C++学习的里程碑,讲解如何构筑单个的类以及类层次结构。你将学习以下专题:

- 面向对象程序设计的基本知识
- 声明基类
- 构造函数
- 析构函数
- 声明类层次
- 虚函数
- 友元函数
- 运算符和友元运算符

11.1 面向对象程序设计的基本知识

我们生活在一个充满对象(object)的世界中。每一个对象都有其属性和操作,有些对象则比其他的更活跃一点。可以将对象按类划分,例如,CASIO Data Bank表就是属于CASIO Data Bank表类的一个对象。

新 词

面向对象的程序设计(object-oriented programming)
面向对象的程序设计运用现实世界的对象概念来开发应用程序。

还可以联想到类层次结构中的类。如CASIO Data Bank表类属于表类层次结构。OOP的基本概念包括类、对象、消息、方法、继承和多态性。

新 词

类(class) 对象(object)
类定义了对象的种属。每个对象都是类的一个实例。

11.2 类和对象

同一类的所有对象具有同一属性和功能。一般地,对象的状态是唯一的,它由其属性的当前值定义。类的功能确定了类的实例所能进行的操作。C++可以调用类数据成员的属性和类成员函数的操作。类将数据成员和成员函数封装起来。

我们回到CASIO表的例子上来,表上的按钮代表CASIO表类的成员函数,显示屏代表数据成员。可以按下某个按钮来编辑日期和/或时间。用OOP术语来说便是,成员函

数通过修改数据成员来改变对象的状态。

11.3 消息和方法

面向对象的程序设计将对象参与的交互动作模型化为事件(event),在事件中消息被发送给对象或在对象之间发送。接收消息的对象调用相应的方法(即 C+ + 中的成员函数)进行响应。C+ + 并不像其他 OOP 语言如 SmalkTalk 那样显式地支持消息和方法的概念。然而,用名词“消息”来讨论对成员函数的调用要容易一些。名词“方法”与“成员函数”是等同的。

新 词

消息(message) **方法**(method)

消息是指对对象做了些什么——what;而方法是指对象如何响应到来的消息——how。

11.4 继 承

在面向对象的语言中,可以从一个类派生出另一个类。

新 词

继承(inheritance)

派生类(也叫子孙类)用继承特性从其父类和祖先类中继承数据成员和成员函数。

类派生通过增加新的属性和新的操作来改进其父类。派生类中一般要声明新的数据成员和新的成员函数。此外,当继承的成员函数的操作不适合派生类时,派生类还可以重写它们。

为了将继承的概念用到 CASIO Data Bank 表上,请考虑以下情景:假定手表厂商想制造一种 CASIO Data Comm 表,它提供与 CASIO Data Bank 表同样的功能,但要增加一个寻呼器。CASIO 的工程师们不用从头开始重新设计新的模型(用 OOP 术语来说,是新的类),而是从现有的 CASIO Data Bank 表入手,在此基础上构思。这一过程也许会是在现有的设计上增加新的属性和操作并修改一些现有操作以适应新设计。因而,CASIO Data Comm 型表集成了 CASIO Data Bank 型表的属性和操作。用 OOP 术语来说,CASIO Data Comm 表类是 CASIO Data Bank 表类的子孙。

11.5 多 态 性

OOP 的多态性使得不同类的实例可对某一消息作出特定的反应(在 C+ + 术语中即函数调用)。例如,在图形(点、线、正方形、矩形、圆、椭圆等)层次结构中,每种图形都有一个 draw 函数负责响应画图请求。

新 词

多态性(polymorphism)

多态性使得不同类的实例能够以适合于各个类的方式对同一函数作出响应。

11.6　声 明 基 类

C+ + 中可以声明封装了数据成员和成员函数的类。这些函数更改与检索数据成员的值并进行一些相关操作。

语　法

基类

声明基类的一般语法是:

```
class className
{
   private:
         < private data members>
         < private constructors>
         < private member functions>
   protected:
         < protected data members>
         < protected constructor>
         < protected member functions>
   public:
         < public data members>
         < public constructors>
         < public destructor>
         < public member functions>
};
```

示　例:

```
class point
{
protected:
      double x;
      double y;
public:
      point(double xVal, double yVal);
      double getX();
      double getY();
      void assign(double xVal, double yVal);
      point& assign(point& pt);
}
```

11.7　类的各成分

从上面讲述的语法可以看出声明时要用到关键字 class。C+ + 类对不同的成员(数据成员和成员函数)提供三种级别的可见性:

- 私有成分
- 保护成分
- 公有成分

新　词

私有成分(private section)　　**保护成分**(protected section)
公有成分(public section)

在私有成分中,只有类的成员函数才能访问私有成员。类的实例不能访问私有成员。在保护成分中,只有类的成员函数及其子孙类可以访问保护成员。类实例不能访问保护成员。在公有成分中指定的类成员对于类的成员函数、类的实例、子孙类的成员函数、以及它们的实例都是可见的。

以下规定适合于类的各个成分:

1. 类的各成分可以任何次序出现。
2. 类的各成分可以出现不止一次。
3. 如果未指明类成分,则 C+ + 编译器将成员看作是保护的。
4. 应尽量避免将数据成员放在公有成分中,除非这种声明可以大大简化设计。数据成员一般应放在保护成分中以允许子孙类成员函数访问它们。
5. 用成员函数设置和查询数据成员的值。用来设置数据成员值的成员函数可以在需要的时候帮助进行确认以及更新其他数据成员。
6. 类中可以有多个构造函数,它们一般位于公有成分中。
7. 类中只能有一个析构函数,它必须在公有成分中进行声明。
8. 成员函数(包括构造函数和析构函数)如果含多条语句则在类声明的外面定义。定义可以位于声明类的同一文件中。

新　词

构造函数(constructor)　　**析构函数**(destructor)

构造函数是必须与宿主类具有同一名字的特殊成员。析构函数自动删除类实例。

第 8 条规定中所指的成员函数的定义在软件的库中一般位于源文件的外面。当定义成员函数时必须用类名描述函数名。这种描述的语法包括后面跟以两个冒号::的类名以及函数名。请看下面的类:

```
class point
{
      protected:
            double x;
            double y;
      public:
            point(double xVal, double yVal);
            double getX();
            // other member functions
};
```

构造函数和成员函数的定义如下:

```
point::point(double xVal, double yVal)
{
  // statements
}

double point::getX()
{
```

```
    // statements
}
```

一旦声明了类,就可以将类名作为类型标识符来声明类实例,其语法与声明变量类似。

我们来看一个例子。清单 11.1 中列出了程序 CLASS1.CPP 的源代码。程序提示键入一个矩形(对象)的长度和宽度,然后显示所指定的矩形的长度、宽度和面积。

键　入　　**清单 11.1　程序 CLASS1.CPP 的源代码**

```
 1: // C++ program that illustrates a class
 2:
 3: #include <iostream.h>
 4:
 5: class rectangle
 6: {
 7:    protected:
 8:       double length;
 9:       double width;
10:    public:
11:       rectangle() { assign(0, 0); }
12:       rectangle(double len, double wide) { assign(len, wide); }
13:       double getLength() { return length; }
14:       double getWidth() { return width; }
15:       double getArea() { return length * width; }
16:       void assign(double len, double wide);
17: }
18:
19: void rectangle::assign(double len, double wide)
20: {
21:    length = len;
22:    width = wide;
23: }
24:
25: main()
26: {
27:    rectangle rect
28:    double len, wide;
29:
30:    cout << "Enter length of rectangle : ";
31:    cin >> len;
32:    cout << "Enter width of rectangle : ";
33:    cin >> wide;
34:    rect.assign(len, wide);
35:    cout << "Rectangle length = " << rect.getLength() << "\n"
36:         << "                wide   = " << rect.getWidth() << "\n"
37:         << "                area   = " << rect.getArea() << "\n";
38:    return 0;
39: }
```

以下是清单 11.1 中程序的一个运行范例:

输　出

```
Enter length of rectangle : 10
Enter width of rectangle : 12
Rectangle length =  10
          width  =  12
          area   =  120
```

分　析

清单 11.1 中的程序声明了对矩形进行模型化的类 rectangle。该类含有两个 double 型的数据成员 length 和 width,它们存放矩形的尺寸。此外,类中还有两个构造函数——缺省的构造函数和非缺省的构造函数。类中还定义了成员函数 getLength, getWidth, getArea 和 assign。

新　词

缺省构造函数(elefault constructor)
非缺省构造函数(nondefault constructor)
缺省构造函数建立一个大小为零的实例,而非缺省构造函数建立一个非零大小的实例。

在类声明中定义的函数 getLength 返回成员 length 的值;函数 getWidth 也是在类声明中定义的,它返回成员 width 的值。在类声明中定义的函数 getArea 返回成员 length 和 width 相乘的结果。

在类声明外面定义的成员函数 assign 将其参数所对应的变元 len 和 wide 分别赋给数据成员 length 和 width。为使函数实现简单一些,这里没有检查负值。

main 函数声明了类实例 rect 以及 double 型的变量 len 和 wide。第 30 行的输出语句提示你输入矩形的长度。第 31 行的语句获取输入并将其存放在变量 len 中。第 32 行的输出语句提示你输入矩形的宽度。第 31 行的语句获取输入并将其存放在变量 wide 中。

函数 main 用 assign 成员函数将输入值赋给实例 rect。用 OOP 术语来说,就是函数 main 向对象 rect 发送了一条赋值消息。此消息的变元是变量 len 和 wide。对象 rect 通过调用方法(成员函数)rectangle::assign(double, double)进行响应。

第 35 到 37 行的输出语句显示对象 rect 的长度、宽度和面积。该语句发送消息 getLength、getWidth、getArea 给对象 rect。对象 rect 依次调用方法(或者说成员函数)以响应每一条消息。

11.8　构 造 函 数

C+ + 构造函数和析构函数都自动地工作以保证类实例的正确建立和删除。

语　法

构造函数

构造函数的一般语法是:

```
class className
{
    public:
        className( ); // default constructor
        className(className& c); // copy constructor
        className( < parameter list> ); // another constructor
};
```

示　例:

```
class point
{
    protected:
        double x;
        double y;
    public:
        point( );
        point( double xVal, double yVal);
        point( point& pt);
        double getX( );
        double getY( );
        void assign(double xVal, double yVal);
        point& assign(point& pt);
};
```

新　词

复制构造函数(copy constructor)

可以用复制构造函数从现有实例中复制数据以建立类实例。

C+ + **中有关构造函数的特性和规定:**

1. **构造函数的名字必须与类名相同。**
2. **不能包含任何返回类型, 甚至** void **也不行。**
3. **一个类中可以有任意数量的构造函数, 甚至可以没有。在后一种情况下, 编译器自动为该类生成一个。**
4. **缺省的构造函数或者不带参数或者参数表中所有参数都使用缺省变元。这里有两个例子:**

```
// class uses parameterless constructor
class point1
{
    protected:
        double x;
        double y;
    public:
        point1( );
        // other member functions
};

// class uses constructor with default arguments
class point2
```

```
{
    protected:
        double x;
        double y;
    public:
        point(double xVal = 0, double yVal = 0);
        // other member functions
};
```

5. 复制构造函数使你可以利用现有实例建立新的类实例。举例如下:

```
class point
{
    protected:
        double x;
        double y;
    public:
        point();
        point(double xVal, double yVal);
        point(point& pt);
        // other member functions
}
```

6. 类实例的声明(包括函数参数和局部实例)要用到构造函数。哪一个构造函数会被调用?答案取决于你为类定义了多少个构造函数以及类实例是如何声明的。例如,我们看看最后一个版本的类 point 的实例:

```
point p1; // involves the default constructor
point p2(1.1, 1.3); // uses the second constructor
point p3(p2); // uses the copy constructor
```

因为实例 p1 未指定变元,编译器使用缺省构造函数。实例 p2 指定两个浮点数变元,因而编译器使用第 2 个构造函数。实例 p3 以实例 p2 为变元,因此编译器使用复制构造函数由实例 p2 建立实例 p3。

要声明复制构造函数,特别是对于那些建立动态数据结构模型的类。这些构造函数进行的复制可包含动态数据,称为深度复制(deep copy)。缺省情况下,编译器建立一个叫作浅复制(shallow copy)的构造函数,它仅仅复制数据成员。

不要指望浅复制构造函数复制那些其成员中含指针的类的实例。

11.9 析 构 函 数

C+ + 类中还可含自动删除类实例的析构函数。

语　法

析构函数

析构函数的一般语法是:

```
class className
{
    public:
        className(); // default constructor
        // other constructors
        ~className()
        // other member function
};
```

示　例:

```
class String
{
    protected:
        char * str;
        int len;

    public:
        String();
        String(String& s);
        ~String();
        // other member functions
};
```

C+ + 中有关析构函数的特性和规定:

1. 析构函数名必须以代字号(~)打头。析构函数名的其余部分必须与类名相同。
2. 不得包含返回类型,甚至 void 也不行。
3. 一个类中只能有一个析构函数。此外,如果省略析构函数,则编译器会自动建立析构函数。
4. 析构函数不得带任何参数。
5. 当出了类实例的作用域后,运行系统会自动调用类的析构函数。

11.10　构造函数和析构函数举例

我们来看一个使用构造函数和析构函数的典型例子。清单 11.2 中列出了 CLASS2.CPP 程序的源代码。该程序完成下列动作:

- 建立一个动态数组(对象);
- 为动态数组的元素赋值;
- 显示动态数组的值;
- 删除动态数组。

键　入　　清单 11.2　程序 CLASS2.CPP 的源代码

```
1: // Program demonstrates constructors and destructors
```

```
 2:
 3: # include < iostream.h>
 4:
 5: const unsigned MIN_SIZE = 4;
 6:
 7: class Array
 8: {
 9:     protected:
10:       double * dataPtr;
11:       unsigned size;
12:
13:     public:
14:       Array(unsigned Size = MIN_SIZE);
15:       ~Array()
16:         { delete [] dataPtr; }
17:       unsigned getSize() const
18:         { return size; }
19:       void store(double x, unsigned index)
20:         { dataPtr[index] = x; }
21:       double recall(unsigned index)
22:         { return dataPtr[index]; }
23: };
24:
25: Array::Array(unsigned Size)
26: {
27:   size = (Size < MIN_SIZE) ? MIN_SIZE : Size;
28:   dataPtr = new double[size];
29: }
30:
31: main()
32: {
33:   Array Arr(10);
34:   double x;
35:   // assign data to array elements
36:   for (unsigned i = 0; i < Arr.getSize(); i++ ) {
37:     x = double(i);
38:     x = x * x - 5 * x + 10;
39:     Arr.store(x, i);
40:   }
41:   // display data in the array element
42:   cout << "Array Arr has the following values:\n\n";
43:   for (i = 0; i < Arr.getSize(); i++ )
44:     cout << "Arr[" << i << "] = " << Arr.recall(i) << "\n";
45:   return 0;
46: }
```

以下是清单 11.2 中程序的一个运行范例:

输　出

```
Array Ar1 has the following values:
```

```
Ar1[0] =  10
Ar1[1] =  6
Ar1[2] =  4
Ar1[3] =  4
Ar1[4] =  6
Ar1[5] =  10
Ar1[6] =  16
Ar1[7] =  24
Ar1[8] =  34
Ar1[9] =  46
```

分　析

清单 11. 2 中的程序在第 5 行声明了常量 MIN- SIZE, 它指定动态数组的最小容量。程序在第 7 行还声明了类 Array。该类有两个数据成员: dataPtr 和 size。成员 dataPtr 是指向元素的动态分配的数组的指针。成员 size 中存储类 Array 的一个实例中的元素个数。

类中声明了缺省的构造函数(构造函数实际上有一个带缺省值 MIN- SIZE 的参数)。程序在第 25 到 29 行定义了构造函数。参数 Size 所对应的变元指定数组元素的个数。第 27 行的语句将 Size 和常量 MIN- SIZE 中较大的值赋给数据成员 size。第 28 行的语句用运算符 new 为数组分配动态空间。该语句将动态数组的基地址赋给成员 dataPtr。

析构函数～Array 对成员 dataPtr 使用运算符 delete 以释放数组的动态空间。

在类声明中定义的成员函数 getSize 返回数据成员 size 的值。

函数 store 也是在类声明中定义的, 它将参数 x 所传递的值存到由参数 index 指定的序号的元素中。为简化函数的实现, 没有进行索引越界检查。

在类声明中定义的函数 recall 返回由参数 index 指定的元素的值。为简化函数的实现, 没有进行索引越界检查。

main 函数中声明了对象 Arr 作为类 Array 的实例。第 33 行的声明指定该实例有 10 个元素。main 函数中还声明了 double 型的变量 x。第 36 到 40 行的 for 循环存储实例 Arr 中的值。该循环用了控制变量 i, 并以 1 为增量从 0 到 Arr. getSize() - 1 进行迭代。循环继续条件发送 getSize 消息给实例 Arr 以获得数组中的元素个数。第 37 和 38 行的语句计算出值来并存到实例 Arr 的一个元素中。第 39 行的语句将消息 store 发送给实例 Arr 并传递变元 x 和 i。对象 Arr 将变量 x 的值存在第 i 个元素中。

第 42 行的输出语句给第 43 和 44 行的 for 循环的输出作注释。该循环使用控制变量 i 并以 1 为增量从 0 到 Arr. getSize() - 1 进行迭代。第 44 行的输出语句通过给实例 Arr 发送消息 recall 以显示该实例的元素。该消息带变元 i。

11.11　声明类层次结构

C+ + 的 OOP 特性的威力就在于可以从现有的类中派生出新的类来。子孙类可继承祖先类(父类、祖父类等)的成员并可以重写继承函数的某些部分。用继承可以在子孙类中进行代码复用。

语　法

派生类

声明派生类的一般语法是:

```
class className : [public] parentClass
{
    < friend classes>

    private:
        < private data members>
        < private constructors>
        < private member functions>
    protected:
        < protected data members>
        < protected constructors>
        < protected member functions>
    public:
        < public data members>
        < public constructors>
        < public destructor>
        < public member functions>

        < friend functions and friend operators>
};
```

示　例:

下面的例子中声明了类 cRectangle **及其子孙类** cBox:

```
class cRectangle
{
    protected:
        double length;
        double width;
    public:
        cRectangle(double len, double wide);
        double getLength() const;
        double getWidth() const;
        double assign(double len, double wide);
        double calcArea();
};

class cBox : public cRectangle
{
    protected:
        double height;

    public:
        cBox(double len, double wide, double height);
        double getHeight() const;
        assign(double len, double wide, double height);
        double calcVolume();
};
```

类的亲缘关系由冒号后跟以可选关键字 public **再加上父类的名字来表示。如果包含关键字** public, **则允许子孙类的实例访问父类和其他祖先类的公有成员。相反,如果不用**

public,则子孙类的实例不可访问祖先类的成员。

子孙类继承其祖先类的数据成员。C+ + 中没有提供删除不想继承的数据成员的机制——只能全盘接受。但是,C+ + 允许你重写继承的成员函数。这一课的稍后会详细讨论这个问题。子孙类中可以声明新的数据成员、新的成员函数、重写成员函数。可以根据类设计的需要将这些成员放在私有、保护或公有部分。

要用带缺省变元的参数减少构造函数的数量。

要用成员函数访问数据成员的值。这些成员函数可以让你控制和确认数据成员的值。

不要将类的所有构造函数都声明为保护的,除非你想强迫客户程序(即用到该类的程序)通过声明带公有构造函数的子孙类以使用该类。

不要在公有部分声明数据成员。

我们来看一个声明了一个小的类层次的例子。清单 11.3 中列出了程序 CLASS3.CPP 的源代码。该程序声明的类包含两种简单的几何形状构成的层次关系:圆和圆柱。程序无需输入,它使用内部数据建立几何形状并显示它们的尺寸、面积和体积。

键　入　　**清单 11.3　程序 CLASS3.CPP 的源代码**

```
// Program that demonstrates a small hierarchy of classes

# include <iostream.h>
# include <math.h>

const double pi = 4 * atan(1);

inline double sqr(double x)
{ return x * x; }

class cCircle
{
   protected:
      double radius;

   public:
      cCircle(double radiusVal = 0) : radius(radiusVal) {}
      void setRadius(double radiusVal)
         { radius = radiusVal; }
      double getRadius() const
         { return radius; }
      double area() const
         { return pi * sqr(radius); }
      void showData();
};

class cCylinder : public cCircle
```

```
28: {
29:    protected:
30:       double height;
31:
32:    public:
33:       cCylinder(double heightVal =  0, double radiusVal =  0)
34:        : height(heightVal), cCircle(radiusVal) {}
35:       void setHeight(double heightVal)
36:          { height =  heightVal; }
37:       double getHeight() const
38:          { return height; }
39:       double area() const
40:          { return 2 *  cCircle::area() +
41:                   2 *  pi *  radius *  height; }
42:       void showData();
43: };
44:
45: void cCircle::showData()
46: {
47:    cout << "Circle radius          =  " << getRadius() << "\n"
48:         << "Circle area            =  " << area() << "\n\n";
49: }
50:
51: void cCylinder:showData()
52: {
53:    cout << "Cylinder radius        =  " << getRadius() << "\n"
54:         << "Cylinder height        =  " << getHeight() << "\n"
55:         << "Cylinder area          =  " << area() << "\n";
56: }
57:
58: main()
59: {
60:    cCircle Circle(1);
61:    cCylinder Cylinder(10, 1);
62:
63:    Circle.showData();
64:    Cylinder.showData();
65:    return 0;
66: }
```

以下是清单 11.3 中程序的一个运行范例:

输　出

```
Circle radius          =  1
Circle area            =  3.14159

Cylinder radius        =  1
Cylinder height        =  10
Cylinder area          =  69.115
```

分　析

清单 11.3 中的程序声明了类 cCircle 和 cCylinder。类 cCircle 建立圆的模型,而类 cCylinder 对圆柱建立模型。

类 cCircle 声明了一个数据成员 radius,用它存放圆的半径。类中还声明了构造函数和一些成员函数。当声明一个类实例时,构造函数为数据成员 radius 赋值。注意构造函数用了一种新的语法初始化成员 radius。函数 setRadius 和 getRadius 分别用来设置和查询成员 radius 的值。函数 area 返回圆的面积。函数 showData 显示类实例的半径和面积。

类 cCylinder 是 cCircle 的一个子孙,它声明了数据成员 height 以存放圆柱的高度。该类继承了成员 radius,用它存放圆柱的半径。类 cCylinder 声明了一个构造函数和几个成员函数。当创建类实例时构造函数为成员 radius 和 height 赋值。注意初始化成员时用到了新语法:成员 height 被初始化,而成员 radius 是以 radiusVal 为变元调用类 cCircle 的构造函数而进行初始化的。函数 getHeight 和 setHeight 分别用来设置和查询成员 height 的值。该类使用继承函数 setRadius 和 getRadius 对继承的成员半径进行操作。函数 area 重写了继承函数 cCircle: : area(),它返回圆柱体的表面积。注意此函数显式地调用了继承函数 cCircle: : area()。函数 showData 显示了类实例的半径、高度和面积。

main 函数中声明了类 cCircle 的实例 Circle 并将圆的半径赋值为 1。此外,函数还声明了类 cCylinder 的实例 Cylinder,并为圆柱的高度和半径分别赋值 10 和 1。接下来,函数发送 showData 消息给实例 Circle 和 Cylinder,这两个对象通过调用各自的成员函数响应此消息。

11.12　虚　函　数

前面已经提到过,多态性是面向对象的程序设计的重要特性。考察下面的简单类和函数 main:

```
# include < iostream.h>
class cA
{
    public:
        double A(double x) { return x * x; }
        double B(double x) { return A(x) / 2; }
};

class cB : public cA
{
    public:
        double A(double x) { return x * x * x; }
};

main()
{
    cB aB;
    cout < < aB.B(3) < < "\n";
```

```
    return 0;
}
```

类 cA 中包含函数 A 和 B,而函数 B 要调用函数 A。类 cB 是类 cA 的子孙类,它继承了函数 B 但重写了函数 A。这样做的目的是为了让继承的函数 cA::B 调用函数 cB::A 以支持多态性。程序的输出是什么? 答案是 4.5 而不是 13.5! 为什么? 原来,编译器用继承函数 cA::B 来分解表达式 aB.B(3),而 cA::B 会调用函数 cA::A。因而函数 cB:A 被撇在一边,程序无法支持多态性。

C+ + 中是通过提供虚函数来支持多态性的。

新　词

虚函数(virtual function)

虚函数是通过将关键字 virtual 放在函数的返回类型前声明的,它只在运行时起作用。

一旦声明了一个函数为虚函数,则在子孙类中只能用虚函数来重写它。这些重写的函数必须具有相同的参数表。虚函数可以重写祖先类中的非虚函数。

语　法

虚函数

声明虚函数的一般语法是:

```
class className1
{
    // member functions
    virtual returnType functionName(< parameter list> );
};

class className2 : public className1
{
    // member functions
    virtual returnType functionName(< parameter list> );
};
```

示　例:

本例说明虚函数如何在类 cA 和 cB 中成功地实现多态性:

```
# include < iostream.h>
class cA
{
    public:
        virtual double A(double x) {return x * x; }
        double B(double x) { return A(x) / 2; }
};

class cB : public cA
{
    public:
        virtual double A(double x) {return x * x * x; }
};
```

```
main()
{
    cB aB;
    cout < <  aB. B(3) < <  "\n";
    return 0;
}
```

本程序可显示正确的结果 13. 5, 因为对继承函数 cA: : B 的调用是在运行时通过对 cB: : A 的调用进行分解的。

> **要使用虚函数——如果可调用函数要实现特定于类的行为。将这样的函数声明为虚函数可确保它能提供与关联类有关的正确响应。**
>
> **不要将成员函数声明为虚函数——一般情况下虚函数需要一些额外开销。**

我们来看一个例子。清单 11. 4 中列出了程序 CLASS4. CPP 的源代码。该程序建立一个正方形和矩形, 并显示其尺寸和面积。不需要输入。

键　入　　**清单 11. 4　程序 CLASS4. CPP 的源代码**

```
 1: // Program that demonstrates virtual functions
 2:
 3: # include < iostream. h>
 4:
 5: class cSquare
 6: {
 7:    protected:
 8:       double length;
 9:
10:    public:
11:       cSquare(double len) { length =  len; }
12:       double getLength() { return length; }
13:       virtual double getWidth() { return length; }
14:       double getArea() { return getLength() *  getWidth(); }
15: }
16:
17: class cRectangle : public cSquare
18: {
19:    protected:
20:       double width;
21:
22:    public:
23:       cRectangle(double len, double wide) :
24:           cSquare(len), width(wide) {}
25:       virtual double getWidth() { return width; }
26: };
27:
28: main()
```

```
29: {
30:     cSquare square(10);
31:     CRectangle rectangle(10, 12);
32:
33:     cout << "Square has length = " << square.getLength() << "\n"
34:          << "       and area   = " << square.getArea() << "\n";
35:     cout << "Rectangle has length = "
36:          << rectangle.getLength() << "\n"
37:          << "           and width = "
38:          << rectangle.getWidth() << "\n"
39:          << "           and area  = "
40:          << rectangle.getArea() << "\n";
41:     return 0;
42: }
```

以下是清单 11.4 中程序的一个运行范例:

输　出

```
Square has length =  10
       and area   =  100
Rectangle has length =  10
           and width  =  12
           and area   =  120
```

分　析

清单 11.4 中的程序声明了类 cSquare 和 cRectangle,它们分别建立正方形和矩形的模型。类 cSquare 中声明了单个的数据成员 length 以存放正方形的长度(亦为宽度)。类中声明了带参数 len 的构造函数,它将变元的值传递给成员 length。类中还声明了函数 getLength, getWidth 和 getArea。函数 getLength 和 getWidth 都返回成员 length 的值。注意类中声明函数 getWidth 为 virtual。函数 getArea 通过调用函数 getLength 和 getWidth 进行计算,返回矩形的面积。之所以选择调用这些函数而不直接使用数据成员 length 的值是为了演示虚函数 getWidth 是如何工作的。

程序中声明的类 cRectangle 是类 cSquare 的子孙类。类 cRectangle 声明了数据成员 width 并继承了成员 length。类中使用这些成员存放矩形的基本尺寸。类的构造函数带有参数 len 和 wide,把它们的值传递给成员 length 和 width。注意该构造函数以 len 为变元调用了构造函数 cSquare,并以参数 wide 的值初始化数据成员 width。

类 cRectangle 声明了虚函数 getWidth,它返回数据成员 width 的值。该类继承了成员函数 getLength 和 getArea,因为它们的实现方法也适合于 cRectangle。

main 函数声明了对象 square 作为类 cSquare 的实例。实例 square 的长度为 10,main 函数还声明了对象 rectangle 作为类 cRectangle 的实例。实例 rectangle 的长度为 10、宽度为 12。

第 33 和 34 行的输出语句显示了实例 square 的长度和面积。该语句发送消息

getLength 和 getArea 给以上实例以获得所要求的值。实例 square 调用函数 getArea，后者又调用函数 cSquare: : getLength 和 cSquare: : getWidth。

第 35 到 40 行的输出语句显示实例 rectangle 的长度、宽度和面积。该语句发送消息 getLength、getWidth 和 getArea 给此实例。实例通过调用继承函数 cSquare: : getLength、虚函数 cRectangle: : getWidth 以及继承函数 cSquare: : getArea 进行响应。后一个函数还调用继承函数 cSquare: : getLength 和虚函数 cRectangle: : getWidth 以正确地计算矩形的面积。

> 要将析构函数声明为虚函数。这可以保证撤消类实例时多态性的实现。此外，特别建议你为每个类声明复制构造函数和赋值运算符。

> 不要忘记，如果合适的话子孙类也可以继承虚函数和析构函数。不需要声明只是简单调用父类的相应成员的外壳函数和析构函数。

11.13　虚函数规则

声明虚函数的规则是“一旦为虚，永远为虚”。换句话说，一旦在类中声明了一个函数为虚函数，则子类中只能用另一虚函数(具有相同的参数表)来改写此虚函数。在子孙类中必须有 virtual 声明。初看起来这条规则似乎束缚了你。这一限制对于支持虚函数但不支持重载函数的面向对象的程序设计语言来说自然是正确的，但对 C+ + 来说，情况比较有趣一些。你可以声明非虚的重载函数，它与虚函数具有相同的名字但参数表不同。但是，不能继承与虚函数名字相同的非虚的成员函数。下面的例子可说明这一点：

```
# include < iostream.h>
class cA
{
public:
  cA() {}
  virtual void foo(char c)
   { cout < <  "virtual cA: : foo() returns "< <  c < <  "\n"; }
};

class cB : public cA
{
  public:
   cB() {}
   void foo(const char*  s)
     { cout < <  "cB: : foo() returns "< <  s < <  '\n'; }
   void foo(int i)
     { cout < <  "cB: : foo() returns "< <  i < <  '\n'; }
   virtual void foo(char c)
     { cout < <  "virtual cB: : foo() returns "< <  c < <  '\n'; }
};

class cC : public cB
```

```
{
  public:
   cC() {}
   void foo(const char* s)
     { cout << "cC::foo() returns " << s << '\n'; }
   void foo(double x)
     { cout << "cC::foo() returns " << x << '\n'; }
   virtual void foo(char c)
     { cout << "virtual cC::foo() returns " << c << '\n'; }
};

main()
{
  int n = 100;
  cA Aobj;
  cB Bobj;
  cC Cobj;

  Aobj.foo('A');
  Bobj.foo('B');
  Bobj.foo(10);
  Bobj.foo("Bobj");
  Cobj.foo('C');
  // if you uncomment the next statement, program does not compile
  // Cobj.foo(n)
  Cobj.foo(144.123);
  Cobj.foo("Cobj");
  return 0;
}
```

这一段代码中声明了三个类: cA, cB 和 cC, 它们形成了线性层次关系。类 cA 中声明函数 foo(char) 为虚函数。类 cB 中也声明了它自己的虚函数 foo(char)。此外, 类 cB 中声明了非虚的重载函数 foo(const char * s) 和 foo(int)。类 cC 是类 B 的子孙类, 它声明了虚函数 foo(char) 和非虚的重载函数 foo(const char *) 和 foo(double)。注意, 类 cC 如果确实需要函数 foo(const char *), 则必须声明它, 因为它不能继承成员函数 cB::foo(const char *)。C++ 在涉及到重载的虚函数时支持不同函数继承模式。main 函数为三个类中的每个类建立实例并包容了成员函数 foo 的不同版本。

11.14　友 元 函 数

C++ 允许成员函数访问类的所有数据成员, 此外, C++ 还支持友元函数进行同样级别的访问。友元函数在类中声明且以关键字 friend 开头。除了使用这一特殊关键字以外, 友元函数与成员函数非常相像, 只是友元函数不能返回其友类的引用, 因为这需要返回自引用* this。但是, 如果在其友类外定义友元函数, 那么在描述函数名时是不需要带类名的。

新　词

友元函数(friend function)

友元函数是可以访问一个或多个类的所有数据成员的普通函数。

语　法

友元函数

友元函数的一般格式为:

```
class className
{
     public:
          className();
          // other constructors

          friend returnType friendFunction(< parameter list> );
};
```

示　例:

```
class String
{
    protected:
         char * str;
         int len;

    public:
          String();
          ～String();
          // other member functions
          friend String& append(String& str1, String& str2);
          friend String& append(const char* str1, String& str2);
          friend String& append(String& str1, const char* str2);
};
```

友元函数可以完成成员函数难以完成甚至不可能完成的任务。

我们来看一个使用友元函数的简单例子。清单 11.5 中列出了程序 CLASS5.CPP 的源代码。程序内部生成两个复数,将它们相加并把结果存到另一个复数中,然后再显示操作数和复数结果。

键　入　　**清单 11.5　程序 CLASS5.CPP 的源代码**

```
 1: // Program that demonstrates friend function
 2:
 3: # include < iostream.h>
 4:
 5: class Complex
 6: {
 7:     protected:
 8:       double x
 9:       double y;
10:
11:     public:
```

```
12:        Complex(double read = 0, double imag = 0);
13:        Complex(Complex& c) { assign(c); }
14:        void assign(Complex& c);
15:        double getReal() const { return x; }
16:        double getImag() const { return y; }
17:        friend Complex add(Complex& c1, Complex& c2);
18: };
19:
20: Complex::Complex(double real, double imag)
21: {
22:    x = real;
23:    y = imag;
24: }
25:
26: void Complex::assign(Complex& c)
27: {
28:    x = c.x;
29:    y = c.y;
30: }
31:
32: Complex add(Complex& c1, Complex& c2)
33: {
34:    Complex result(c1);
35:
36:    result.x += c2.x;
37:    result.y += c2.y;
38:    return result;
39: }
40:
41: main()
42: {
43:    Complex c1(2,3);
44:    Complex c2(5,7);
45:    Complex c3;
46:
47:    c3.assign(add(c1,c2));
48:    cout << "(" << c1.getReal() << "+ i" << c1.getImag() << ") "
49:         << "+ "
50:         << "(" << c2.getReal() << "+ i" << c2.getImag() << ") "
51:         << "= "
52:         << "(" << c3.getReal() << "+ i" << c3.getImag() << ") "
53:         << "\n\n";
54:    return 0;
55: }
```

以下是清单 11.5 中程序的一个运行范例:

输　出

```
(2 + i3) + (5 + i7) = (7 + i10)
```

分　析

清单 11. 5 中的程序声明了复数的模型类 Complex。类中声明了两个数据成员、两个构造函数、一个友元函数(本例的重点所在),以及一组成员函数。数据成员 x 和 y 分别存放复数的实部和虚部。

类中有两个构造函数。第 1 个构造函数有两个参数(带缺省变元),它使你可以用复数的实部和虚部建立类实例。因为两个参数都有缺省变元,此构造函数同时也是缺省构造函数。第 2 个构造函数 complex(complex&)是一复制构造函数。

类 Complex 声明了三个成员函数。函数 assign 将类实例复制到另一个实例中。函数 getReal 和 getImg 分别返回成员 real 和 imag 的值。

类 Complex 中声明了友元函数 add 以便将两个复数相加。为使程序简短,没有补充进行类实例的减法、乘法和除法的友元函数。友元函数 add 有何特殊之处? 为什么不用普通成员函数将类实例相加呢? 若声明作为替换的 add 成员函数可回答这一问题:

```
complex& add(complex& c)
```

此声明表明函数将参数 c 作为第 2 个操作数。以下示意成员函数 add 如何工作:

```
complex c1(3, 4), c2(1.2, 4.5);
c1.add(c2); // adds c2 to c1
```

首先,成员函数 add 是以增量函数而不是加法函数方式工作。其次,目标类实例总是第 1 操作数。这对于加法和乘法一类的运算不成问题,但对于减法和除法就很成问题了。而友元函数 add 可以让你自由选择如何将类实例相加,用它可以工作得更好。

友元函数 add 返回的是类实例。该函数建立类 Complex 的一个局部实例并返回该实例。

函数 main 用成员函数 assign 和友元函数 add 进行简单的复数运算。此外,main 函数中还以类 Complex 的不同实例调用了函数 getReal 和 getImg 以显示每个实例的各部分。

11.15　运算符和友元运算符

上一个程序用了成员函数和友元函数实施复杂的数学运算。这种方法在 C 和 Pascal 中很典型,因为这些语言不支持用户定义的运算符。而 C+ + 则不同,它允许你声明运算符和友元运算符,这些运算符包括:

```
+     -     *     /     %     = =     ! =     < =     <     > =     >
+ =     - =     * =     / =     % =     [ ]     ( )     < <     > >
```

有关使用这些运算符的更详细的规定请查阅 C+ + 语言的参考书。C+ + 将运算符和友元运算符作为成员函数和友元函数对待。

语　法

运算符和友元函数

声明运算符和友元运算符的一般语法是:

```
class className
{
    public:
        // constructors and destructor
        // member functions

        // unary operator
        returnType operator operatorSymbol(operand);
        // binary operator
        returnType operator operatorSymbol(firstOperand,
                                           secondOperand);
        // unary friend operator
        friend returnType operator operatorSymbol(operand);
        // binary operator
        friend returnType operator operatorSymbol(firstOperand,
                                                  secondOperand);
};
```

示　例:

```
class String
{
    protected:
        char * str;
        int len;

    public:
        String();
        ~String();
        // other member functions
        // assignment operator
        String& operator = (String& s);
        String& operator += (String& s);
        // concatenation operators
        friend String& operator + (String& s1, String& s2);
        friend String& operator + (const char* s1, String& s2);
        friend String& operator + (String& s1, const char* s2);
        // relational operators
        friend int operator > (String& s1, String& s2);
        friend int operator >= (String& s1, String& s2);
        friend int operator < (String& s1, String& s2);
        friend int operator <= (String& s1, String& s2);
        friend int operator == (String& s1, String& s2);
        friend int operator != (String& s1, String& s2);
};
```

这样你所编写的函数可以像使用预定义运算符一样使用运算符和友元运算符。因此，可以建立运算符支持诸如复数、串、数组和矩阵的模型类的运算。用这些运算符写出的表达式比使用命名函数的表达式具有更强的可读性。

我们来看一个例子。清单 11.6 中列出了 CLASS6.CPP 程序的源代码。此程序是通过修改和扩展清单 11.5 中的程序得到的。新程序进行了更多的操作并显示两组操作数和结果。

键　入　　清单 11.6　程序 CLASS6.CPP 的源代码

```
// Program that demonstrates operators and friend operators

# include < iostream.h>

class Complex
{
    protected:
      double x;
      double y;

    public:
      Complex(double real = 0, double imag = 0)
        { assign(real, imag); }
      Complex(Complex& c);
      void assign(double real = 0, double imag = 0);
      double getReal() const { return x; }
      double getImag() const { return y; }
      Complex& operator = (Complex& c);
      Complex& operator + = (Complex& c);
      friend Complex operator + (Complex& c1, Complex& c2);
      friend ostream& operator < < (ostream& os, Complex& c);
};

Complex::Complex(Complex& c)
{
   x = c.x;
   y = c.y;
}

void Complex::assign(double real, double imag)
{
   x = real;
   y = imag;
}

Complex& Complex::operator = (Complex& c)
{
   x = c.x;
   y = c.y;
   return * this;
}

Complex& Complex::operator + = (Complex& c)
{
   x + = c.x;
   y + = c.y;
   return * this;
}

Complex operator + (Complex& c1, Complex& c2)
```

```
51: {
52:     Complex result(c1);
53:
54:     result.x + = c2.x;
55:     result.y + = c2.y;
56:     return result;
57: }
58:
59: ostream& operator < < (ostream& os, Complex& c)
60: {
61:    os < < "(" < < c.x < < "+ i" < < c.y < < ") ";
62:    return os;
63: }
64:
65: main( )
66: {
67:    Complex c1(3, 5);
68:    Complex c2(7, 5);
69:    Complex c3;
70:    Complex c4(2, 3);
71:
72:    c3 = c1 + c2;
73:    cout < < c1 < < "+ " < < c2 < < "= " < < c3 < < "\n";
74:    cout < < c3 < < "+ " < < c4 < < "= ";
75:    c3 + = c4;
76:    cout < < c3 < < "\n";
77:    return 0;
78: }
```

这里给出清单 11.6 中程序的一个运行范例:

输　出

```
(3 + i5) + (7 + i5) = (10 + i10)
(10 + i10) + (2 + i3) = (12 + i13)
```

分　析

新的 Complex 类用运算符= 替换了成员函数 assign(Complex&)。新类还用友元运算符+ 替换了友元函数 add:

```
Complex& operator = (Complex& c);
friend Complex operator + (Complex& c1, Complex& c2);
```

运算符= 有一个参数,该参数是类 Complex 的实例的引用,运算符返回对同一类的引用。友元运算符+ 有两个参数(两个都是对类 Complex 实例的引用),它产生复数类类型。

程序中还增加了两个新的运算符:

```
complex& operator + = (complex& c);
friend ostream& operator < < (ostream& os, complex& c);
```

运算符+ = 是类 Complex 的成员。它带一个参数,该参数是类 Complex 实例的引用,运算符产生对同一类的引用。另一新运算符是友元运算符< < ,它示意如何为类编写流提取运算符。该友元运算符带两个参数: 对类 ostream 的引用(输出流类)和对类 Complex 的引用。运算符< < 返回对类 ostream 的引用。这种类型的值使你可以将流输出和其他预定义类型或其他类串在一起(假定这些类也有友元运算符< <)。友元运算符< < 的定义包含有两个语句。第 1 个将串和类 Complex 的数据成员输出到输出流参数 os 中。运算符< < 的友元性使得它可以访问其 Complex 型参数 c 的数据成员 real 和 imag。运算符定义中的第 2 个语句返回其第 1 个参数 os。

函数 main 中声明了类 Complex 的 4 个实例: c1, c2, c3 和 c4。实例 c1, c2 和 c4 是通过将非缺省值赋给数据成员 real 和 imag 而建立的。函数用运算符= , + , < < , + = 进行测试。该程序表明可以用运算符和友元运算符编写更具可读性的代码并可支持更高层次的抽象。

小　　结

今天这一课介绍了 C+ + 类并讨论了以下内容:

- 面向对象的程序设计的基本概念包括类、对象、消息、方法、继承和多态性。
- 声明基本类可指定私有、保护和公有成分。C+ + 类中包括数据成员和成员函数。数据成员中存储类实例的状态,成员函数对这些状态进行查询和操作。
- 构造函数和析构函数支持对类实例的自动建立和删除。构造函数是与宿主类具有相同名字的特殊成员。可以声明任意多个构造函数,或者没有构造函数。后一种情况下,编译器将自动生成一个构造函数。每一构造函数使你可以用一种不同的方法建立一个类实例。有两种特殊的构造函数: 缺省构造函数和复制构造函数。与构造函数不同, C+ + 只允许你声明一个不带参数的析构函数。析构函数自动删除类实例。当进入或退出类实例的作用域时,运行系统自动调用构造函数和析构函数。
- 声明类层次使你可以从现有类中进行类派生。子孙类继承其祖先类的成员。C+ + 类可以通过定义自己的函数版本以重写继承的成员函数。如果重写非虚函数,则可用不同的参数表声明新版本。相反,不能改变继承的虚函数的参数表。
- 虚成员函数使类可以支持多态性。这种特性可以提供适合于层次结构中每一个类的响应。一旦声明了函数为虚函数,则在子孙类中只能用虚函数重写它。类层次中一个虚函数的所有版本必须具有相同的特征。
- 友元函数是可以访问保护和私有数据成员的特殊的非成员函数。这些函数使你可以实施比成员函数所能提供的更加灵活的操作。
- 运算符和友元运算符可以支持各种运算,例如加法、赋值和指数。可以用这些运算符对类进行某种程度的抽象。此外,它们有助于使操作类实例的表达式更具可读性、更直观。

问　与　答

问: 如果将缺省、复制和其他构造函数声明为"保护的"会发生什么情况?

答: 客户程序无法建立此类的实例。但是,客户程序可以通过声明带公有构造函数的子孙类来使用此类。

问: 可以用构造函数进行类型强制吗?

答: 是的,可以将这种类型强制与类实例的建立结合起来。例如,如果类 Complex 具有构造函数 Complex(double real, double imag),则可以声明类 Complex 的实例 c 如下:

```
Complex c = Complex(1.7, 2.4);
```

问: 可以将消息串在实例上吗?

答: 可以,只要被串起来的消息调用的成员函数返回对接收消息的同一类的引用即可。例如,如果类 String 中有下列成员函数:

```
String& upperCase();
String& reverse();
String& mapChars(char find, char replace);
```

对类 String 的实例 s 写出如下的语句:

```
s.upperCase().reverse().mapChar('', '+');
```

问: 如果类依赖用编译器生成的复制构造函数来复制带有指针的类的实例,那会发生什么情况?

答: 这些构造函数进行的是位(bit by bit)复制。因而,其结果是两个类实例中相应的指针成员具有同一动态数据的地址。这种复制是产生祸害的根源!

问: 可以建立实例数组吗?

答: 可以,但对应的类必须有缺省构造函数,数组实例使用此构造函数。

问: 可以用指针建立类实例吗?

答: 可以,这要用到运算符 new 和 delete 为实例分配和撤消动态空间。以下是使用类 Complex 的一个例子:

```
Complex* pC;
pC = new Complex;
// manipulate the instance accessed by pointer pC
delete pC;
```

或

```
Complex* pC = new Complex;
// manipulate the instance accessed by pointer pC
delete pC;
```

作　　业

作业中所提供的测验题帮助你巩固所学内容,而练习则就使用所学内容提供一些经验。要弄懂测验和练习的解答后再继续下一天的课程。附录 B 的“答案”中提供了解答。

测　验

1. 下面的类声明中有何错误?

```
class String {
        char * str;
        unsigned len;
        String();
        String(String& s);
        String(unsigned size, char = '');
        String(unsigned size);
        String& assign(String& s);
        ~String();
        unsigned getLen() const;
        char* getString();
        // other member functions
};
```

2. 下面的类声明中有何错误?

```
class String {
      protected:
        char * str;
        unsigned len;
      public:
        String();
        String(const char* s);
        String(String& s);
        String(unsigned size, char = '');
        String(unsigned size);
        ~String();
        // other member functions
};
```

3. 以下语句是否正确?它在上面的 String 类声明的基础上建立实例 s:

```
s = String("Hello Visual C+ + ");
```

4. 阅读程序 CLASS6.CPP,如果将函数 main 中实例的声明修改如下,那么程序是否仍能编译?

```
Complex c1 = Complex(3, 5);
Complex c2 = Complex(7, 5);
Complex c3 = c1;
Complex c4 = Complex(2, 3);
```

练　习

根据程序 CLASS6.CPP **建立一个新的程序** CLASS7.CPP,**将单个实例** c1～c4 **换成实例数组** c。

第12天

基本流文件I/O

今天这一课讲解用C++流库进行文件I/O操作。虽然C中的STDIO.H库已经由ANSI C委员会进行了标准化,但C++流库却没有。你可以选择使用STDIO.H中的或者C++流库中的文件I/O函数。这两个I/O库功能都很强、很灵活。学习了今天这一课介绍的基本的实用操作以后就可以从文件中读取数据或将数据写到文件中了。你将学习如下专题:

- 通用流I/O函数
- 顺序文本流I/O
- 顺序二进制数据流I/O
- 随机存取二进制数据流I/O

有关C++流库的更详细内容,请查阅C++语言参考书,如Tom Swan的C++ Primer(SAMS 1992)。

12.1 C++流库

C++流I/O库是由声明于数个头文件中的具有层次关系的类组成的。本书前面所用的IOSTREAM.H只不过是其中的一个。其他文件包括IO.H, ISTREAM.H, OSTREAM.H, IFSTREAM.H, OFSTREAM.H和FSTREAM.H。IO.H头文件中声明了低层次的类和标识符。ISTREAM.H和OSTREAM.H文件支持基本的输入和输出流类。IOSTREAM.H将前面两个头文件中的类操作合并起来。类似地,IFSTREAM.H和OFSTREAM.H文件支持基本文件输入和输出流类。FSTREAM.H文件将前面两个头文件中的类的操作合并起来。另外还有一些流库文件可提供更专门的流I/O。C++ ANSI委员会将定义标准流I/O库,这样就再也不会搞不清楚哪些类和头文件是标准的、哪些不是。

12.2 通用流I/O函数

这一节将介绍对顺序和随机访问I/O都通用的流I/O函数。这些函数包括open, close, good和fail,还有运算符"!"。

用open函数可以打开流文件以进行输入、输出、添加以及既输入又输出。函数中还可以指定有关I/O是二进制的还是文本的。

语　法

open 函数

open 函数的原型是:

```
void open(const char* filename,
          int mode,
          int m = filebuf::openprot);
```

参数 filename 指要打开的文件名; 参数 mode 标识 I/O 方式。以下是参数 mode 的取值表, 这些值是在 IO.H 头文件中定义的:

in	打开输入流
out	打开输出流
ate	将流指针移到文件末尾
app	打开添加流
trunc	如果文件已存在, 则将其大小截为 0
nocreate	如果文件不存在, 则出错
noreplace	如果文件已存在, 则出错
binary	以二进制方式打开

示　例:

```
// open stream for input
fstream f;
f.open("\\AUTOEXEC.BAT", ios::in);

// open stream for output
fstream f;
f.open("\\AUTOEXEC.OLD", ios::out);

// open stream for binary input and output
fstream f;
f.open("INCOME.DAT", ios::in ©¦ios::out ©¦ios::binary);
```

注　意　文件流类中提供了包含函数 open 的动作的构造函数(具有相同参数)。

函数 close 关闭流并恢复有关资源。这些资源包括流 I/O 操作所用到的内存。

语　法

close 函数

close 函数的原型是:

```
void close();
```

示　例:

```
fstream f;
// open stream
f.open("\\AUTOEXEC.BAT", ios::in);
// process file
// now close stream
f.close();
```

C++ 流库包括一组用来检查流操作的出错状态的基本函数。这些函数包括:

1. good()函数,如果流操作未出错则返回非零值。函数 good 声明如下:
 int good();
2. fail()函数,如果流操作出错则返回非零值。函数 fail 声明如下:
 int fail();
3. 对流实例运用重载运算符"!"以确定错误的状态。

C++ 流库还提供了另一些函数以设置并查询流错误的类型以及其他方面的内容。

12.3　顺序文本流 I/O

顺序文本 I/O 所用到的函数和运算符很简单——在前面几课中你已见识过其中的大多数。函数和运算符包括以下这些:

1. 流提取符"<<"将串和字符写到流中;
2. 流插入符">>"从流中读取字符;
3. getline 函数从流中读取串。

语　法

getline 函数

getline 函数的原型如下:

```
istream& getline(signed char* buffer,
                 int size,
                 char delimiter = '\n');
istream& getline(unsigned char* buffer,
                 int size,
                 char delimiter = '\n');
```

参数 buffer 是指向从流中接收到的字符串的指针;参数 size 指定最多要读取多少个字符;参数 delimiter 则指定一定界符,在输入串达到由参数 size 指定的长度以前如果遇到该字符则输入终止。参数 delimiter 的缺省值是'\n'。

示　例:

```
fstream f;
char textline[MAX];
f.open("\\CONFIG.SYS", ios::in);
while (! f.eof()) {
  f.getline(textline, MAX);
  cout << textLine << "\n");
}
f.close()
```

我们来看一个例子。清单 12.1 为程序 IO1.CPP 的源代码。该程序完成以下操作:

- 提示你键入供输入用的文本文件的名字;
- 提示你键入供输出用的文本文件的名字(程序会检查输入和输出文件名是否相同,如果相同,则提示输入一个不同的输出文件名);

· 从输入文件中读取各行并删除这些行后面的空格;
· 将这些行写到输出文件中并显示在标准输出窗口中。

键　入　　**清单 12.1　程序 IO1.CPP 的源代码**

```
1: // C++ program that demonstrates sequential file I/O
2:
3: #include <iostream.h>
4: #include <fstream.h>
5: #include <string.h>
6:
7: enum boolean { false, true };
8:
9: const unsigned LINE_SIZE = 128;
10: const unsigned NAME_SIZE = 64;
11:
12: void trimStr(char* s)
13: {
14: int i = strlen(s) - 1;
15: // locate the character where the trailing spaces begin
16: while (i >= 0 && s[i] == ' ')
17:     i--;
18: // truncate string
19: s[i+1] = '\0';
20: }
21:
22: void getInputFilename(char* inFile, fstream& f)
23: {
24: boolean ok;
25:
26: do {
27:     ok = true;
28:     cout << "Enter input file: ";
29:     cin.getline(inFile, NAME_SIZE);
30:     f.open(inFile, ios::in);
31:     if (!f) {
32:       cout << "Cannot open file " << inFile << "\n\n";
33:       ok = false;
34:     }
35: } while (!ok);
36:
37: }
38:
39: void getOutputFilename(char* outFile, const char* inFile,
40:                        fstream& f)
41: {
42: boolean ok;
43:
44: do {
45:     ok = true;
```

```
46:       cout << "Enter output file: ";
47:       cin.getline(outFile, NAME_SIZE);
48:       if (stricmp(inFile, outFile) != 0) {
49:           f.open(outFile, ios::output);
50:           if (!f) {
51:            cout << "File" << outFile << " is invalid\n\n";
52:            ok = false;
53:           }
54:       }
55:       else {
56:           cout << "Input and output files must be different! \n";
57:           ok = false;
58:       }
59:      } while (!ok);
60: }
61:
62: void processLines(fstream& fin, fstream& fout)
63: {
64:    char line(LINE_SIZE + 1);
65:
66:    // loop to trim trailing spaces
67:    while (fin.getline(line, LINE_SIZE)) {
68:      trimStr(line);
69:      // write line to the output file
70:      fout << line << "\n";
71:      // echo updated line to the output window
72:      cout << line << "\n";
73:    }
74:
75: }
76: main()
77: {
78:
79:      fstream fin, fout;
80:      char inFile[NAME_SIZE + 1], outFile[NAME_SIZE + 1];
81:
82:      getInputFilename(inFile, fin);
83:      getOutputFilename(outFile, inFile, fout);
84:      processLines(fin, fout);
85:      // close streams
86:      fin.close();
87:      fout.close();
88:      return 0;
89: }
```

以下是清单 12.1 中程序的一个运行范例:

输　出

```
Enter input file: sample.txt
```

```
Enter output file： sample. out
This is line 1
This is line 2
This is line 3
This is line 4
```

分　析

清单 12. 1 中的程序并没有声明类而是将焦点集中到用文件流输入和输出文本上。程序声明了函数 trimStr，getInputFilename，getOutputFilename，processLines 和 main。

函数 trimStr 截去了参数 s 中所传递的字符串尾部的空格。函数声明了局部变量 i 并将其赋值为空终结符前的一个字符的索引。函数在第 13 行用 while 循环反向扫描串 s 中的字符以找出第一个非空格字符。第 16 行的语句将空终结符赋给串 s 中的最后一个非空格字符的后一个字符。

函数 getInputFilename 获取输入文件名并打开相应的输入文件流。参数 inFile 将输入文件的名字传递给函数 caller。引用参数 f 将已打开的输入流传递给函数 caller。函数 getInputFilename 声明了局部标志 ok。该函数在第 26 到 35 行中用了 do-while 循环以获取一有效的文件名并打开该文件以用于输入。第 27 行含循环内的第 1 条语句，它将枚举值 true 赋给局部变量 ok。第 28 行的输出语句提示输入文件名。第 29 行语句调用流输入函数 getline 获取输入并将其存在参数 inFile 中。第 30 行语句用流参数 f 打开输入文件。打开语句用 ios::in 值指示打开流用于文本输入。第 31 行的 if 语句确定流 f 是否已成功打开。如果没有成功，函数执行第 32 和 33 行的语句。这些语句显示错误消息并将枚举值 false 赋给局部变量 ok。第 35 行中，循环体的 while 语句检查条件! ok。循环一直迭代到你提供了一个有效的文件名且该文件被成功地打开以用于输入。

函数 getOutputFilename 对函数 getInputFilename 进行补充并带有三个参数。参数 outFile 将输出文件名传递给函数 caller。参数 inFile 为函数提供输入文件名。函数用此参数确保输入和输出文件名不相同。参数 f 将输出流传递给函数 caller。函数 getOutputFilename 的实现与函数 getInputFilename 非常类似。主要区别在于函数 getOutputFilename 调用了函数 stricmp 比较参数 inFile 和 outFile 的值。函数用 stricmp 的结果确定输入和输出文件名是否相同。如果相同，函数执行第 57 和 58 行中的 else 子句。这些子句显示错误消息并将 false 赋给局部变量 ok。

函数 processLines 从输入文件流中读取各行，对其进行修剪，并将它们写到输出文件流中。参数 fin 和 fout 分别传递输入和输出文件流。函数声明了局部串变量 line 并在第 67 到 73 行使用 while 循环处理文本行。while 语句中调用了函数 getline，它读取输入流的下一行并将输入行赋给变量 line。如果已经没有输入行，则 getline 的返回值使 while 循环停止迭代。第 68 行，循环内的第 1 条语句以 line 为变元调用函数 trimStr。该函数调用删去局部变量 line 中的所有尾部空格。第 70 行的语句将变量 line 中的串写到输出文件流中。第 72 行的语句将 line 中的串回显到标准输出窗口中。将这条语句放在程序中的目的是为了便于监控程序的运行。

函数 main 中声明了文件流 fin 和 fout、以及串变量 inFile 和 outFile。第 82 行的语句

以 inFile 和 fin 为变元调用函数 getInputFilename。此调用分别通过变元 inFile 和 fin 获取输入文件名和输入流。第 83 行语句以 outFile, inFile 和 fout 为变元调用函数 getOutputFilename。此调用分别通过变元 outFile 和 fout 获取输出文件名和输出流。第 84 行语句以 fin 和 fout 为变元调用函数 processLines。此调用处理输入文件流 fin 中的各行并将结果写到输出文件流 fout 中。第 86 和 87 行的语句分别关闭输入和输出文件流。

12.4　顺序二进制文件流 I/O

C++ 流库提供了重载的流函数 write 和 read 以用于顺序二进制文件流 I/O。函数 write 向输出流发送数个字节。该函数可以将任意变量或实例写到流中。

语　法

write 函数

重载函数 write 的原型是:

```
ostream& write(const signed char* buff, int num);
ostream& write(const unsigned char* buff, int num);
```

参数 buff 是指向包含要发送到输出流的数据的缓冲区的指针。参数 num 指示要发送到流的缓冲区中的字节数。

示　例:

```
const MAX = 80;
char buff[MAX+1] = "Hello World!";
int len = strlen(buffer) + 1;
fstream f;
f.open("CALC.DAT", ios::out ©¦ios::binary);
f.write((const unsigned char* )&len, sizeof(len));
f.write((const unsigned char* )buff, len);
f.close();
```

函数 read 从输入流中接收数个字节。该函数可以从流中读取任意变量。

语　法

read 函数

重载函数 read 的原型是:

```
istream& read(signed char* buff, int num);
istream& read(unsigned char* buff, int num);
```

参数 buff 是指向从输入流接收输入的缓冲区的指针; 参数 num 表示从流中读取的字节数。

示　例:

```
const MAX = 80;
char buff[MAX+1];
int len;
fstream f;
f.open("CALC.DAT", ios::in ©¦ios::binary);
```

```
f.read((const unsigned char* )&len, sizeof(len));
f.read((const unsigned char* )buff, len);
f.close();
```

我们来看一个进行顺序二进制流 I/O 的例子。清单 12.2 中列出了程序 IO2.CPP 的源代码。程序声明了建立动态数值数组模型的类。流 I/O 操作使得程序可以读写二进制文件中的单个数组元素或整个数组。程序建立了数组 arr1, arr2 和 arr3, 并完成以下操作:

- 为数组 arr1(该数组中有 10 个元素)中的元素赋值;
- 为数组 arr3(该数组中有 20 个元素)中的元素赋值;
- 显示数组 arr1 中的值;
- 将数组 arr1 中的元素写到文件 ARR1.DAT 中去, 每次写一个元素;
- 将数组 arr1 的值从文件读到数组 arr2(arr2 中有 10 个元素——与数组 arr1 一样大)中;
- 显示数组 arr2 的值;
- 显示数组 arr3 的值;
- 将数组 arr3 的值一次全部写到文件 ARR3.DAT 中;
- 一次读出文件 ARR3.DAT 中的数据, 并将其存到数组 arr1 中;
- 显示数组 arr1 的值(输出表明数组 arr1 与数组 arr3 具有相同的大小和数据)。

键　入　　清单 12.2　程序 I02.CPP 的源代码

```
1: /*
2: C++ program that demonstrates sequential binary file I/O
3: */
4:
5: #include <iostream.h>
6: #include <fstream.h>
7:
8: const unsigned MIN_SIZE = 10
9: const double BAD_VALUE = -1.0e+30
10: enum boolean { false, true };
11:
12: class Array
13: {
14: protected:
15:     double *dataPtr;
16:     unsigned size;
17:     double badIndex;
18:
19: public:
20:     Array(unsigned Size = MIN_SIZE);
21:     ~Array()
22:       { delete [] dataPtr; }
23:     unsigned getSize() const { return size; }
```

```
24:     double& operator [](unsigned index);
25:     { return (index < size) ? *(dataPtr + index) : badIndex; }
26:     boolean writeElem(fstream& os, unsigned index);
27:     boolean readElem(fstream& is, unsigned index);
28:     boolean writeArray(const char* filename);
29:     boolean readArray(const char* filename);
30: };
31:
32: Array::Array(unsigned Size)
33: {
34:   size = (Size < MIN_SIZE) ? MIN_SIZE : Size;
35:   badIndex = BAD_VALUE;
36:   dataPtr = new double[size];
37: }
38:
39: boolean Array::writeElem(fstream& os, unsigned index)
40: {
41:     if (index < size) {
42:     os.write((unsigned char*)(dataPtr + index), sizeof(double));
43:       return (os.good()) ? true : false;
44: }
45: else
46:    return false;
47: }
48:
49: boolean Array::readElem(fstream& is, unsigned index)
50: {
51:     if (index < size) {
52:     is.read((unsigned char*)(dataPtr + index), sizeof(double));
53:     return (is.good()) ? true : false;
54:   }
55:   else
56:     return false;
57: }
58:
59: boolean Array::writeArray(const char* filename)
60: {
61:     fstream f(filename, ios::out ©¦ios::binary);
62:
63:     if (f.fail())
64:       return false;
65:     f.write((unsigned char*)&size, sizeof(size));
66:     f.write((unsigned char*)dataPtr, size* sizeof(double));
67:     f.close();
68:     return (f.good()) ? true : false;
69: }
70:
71: boolean Array::readArray(const char* filename)
72: {
73:   fstream f(filename, ios::in ©¦ios::binary);
74:   unsigned sz;
75:
```

```
76:    if (f.fail)
77:       return false;
78:       f.read((unsigned char* ) &sz, sizeof(sz));
79:       // need to expand the array
80:       if (sz ! = size) {
81:          delete [] dataPtr;
82:          dataPtr = new double[sz];
83:          size = sz;
84: }
85:       f.read((unsigned char* )dataPtr, size * sizeof(double));
86:       f.close();
87:       return (f.good()) ? true : false;
88:  }
89:
90:  main()
91:  {
92:       const unsigned SIZE1 = 10;
93:       const unsigned SIZE2 = 20;
94:       char* filename1 = "array1.dat";
95:       char* filename2 = "array3.dat";
96:       Array arr1(SIZE1), arr2(SIZE1), arr3(SIZE2);
97:       fstream f(filename1, ios::out ©¦ios::binary);
98:
99:       // assign values to array arr1
100:       for (unsigned i = 0; i < arr1.getSize(); i++ )
101:       arr1[i] = 10 + i;
102:
103:       // assign values to array arr3
104:    for (i = 0; i < SIZE2, i++ )
105:       arr3[i] = i;
106:
107:    cout < < "Array arr1 has the following values:\n";
108:    for (i = 0; i < arr1.getSize(); i++ )
109:       cout < < arr1[i] < < " ";
110:    cout < < "\n\n";
111:
112:    // write elements of array arr1 to the stream
113:    for (i = 0; i < arr1.getSize(); i++ )
114:       arr1.writeElem(f, i);
115:    f.close();
116:
117:    // reopen the stream for input
118:    f.open(filename1, ios::in ©¦ios::binary);
119:
120:    for (i = 0; i < arr1.getSize(); i + + )
121:       arr2.readElem(f, i);
122:    f.close();
123:
124:    // display the elements of array arr2
125:    cout < < "Array arr2 has the following values:\n";
126:    for (i = 0; i < arr2.getSize(); i++ )
127:       cout < < arr2[i] < < " ";
```

```
128:    cout << "\n\n";
129:
130:    // display the elements of arr3
131:    cout << "Array arr3 has the following values:\n";
132:    for (i = 0; i < arr3.getSize(); i++)
133:       cout << arr3[i] << " ";
134:    cout << "\n\n"
135:
136:    // write the array arr3 to file ARRAY3.DAT
137:    arr3.writeArray(filename2);
138:    // read the Array arr1 from file ARR3.DAT
139:    arr1.readArray(filename2);
140:
141:    // display the elements of array arr1
142:    cout << "Array arr1 now has the following values:\n";
143:    for (i = 0; i < arr1.getSize(); i++)
144:       cout << arr1[i] << " ";
145:    cout << "\n\n";
146:    return 0;
147:
```

以下是清单 12.2 中程序的一个运行范例:

输　出

```
Array arr1 has the following values:
0   10   20   30   40   50   60   70   80   90

Array arr2 has the following values:
0   10   20   30   40   50   60   70   80   90

Array arr3 has the following values:
0   1   2   3   4   5   6   7   8   9   10   11   12   13   14   15   16   17   18   19

Array arr1 now has the following values:
0   1   2   3   4   5   6   7   8   9   10   11   12   13   14   15   16   17   18   19
```

分　析

清单 12.2 中的程序声明的类 Array 与第 11 天中的清单 11.2 中的类似。主要区别在于这次用了运算符[]代替成员函数 store 和 recall。该运算符检查索引的合法性,如果越界则返回成员 badIndex 中的值。除了运算符[]之外,还增加了成员函数 writeElem, readElem, writeArray 和 readArray 以进行顺序二进制文件流 I/O 操作。

第 39 到 47 行定义了函数 writeElem,它将一个数组元素写到输出流中。参数 os 代表输出流。参数 index 指定要写的数组元素。函数 writeElem 在 index 值合法且流输出未出错时返回 true。writeElem 每写完一个数组元素,内部流指针就向前进到下一个位置。

第 49 到 57 行定义了函数 readElem,它从输入流中读取一个数组元素。参数 is 代表

输入流。参数 index 指定要读的数组元素。函数 readElem 在 index 值合法且流输入未出错时返回 true。readElem 每读出一个数组元素,内部流指针就向前进到下一个位置。

函数 writeElem 允许同一类实例向多个流写数据元素,同样,函数 readElem 允许同一类实例从多个流中读取数据。

第 59 到 69 行定义了函数 writeArray,它将整个数组的元素写到二进制文件中。参数 filename 指定输出文件名。函数打开一输出流写入数据成员 size 的值并将动态数组的元素写入。如果函数 writeArray 成功地将数组写到流中则返回 true,否则函数将返回 false。该函数用流函数 open 带文件名和 I/O 方式变元打开一逻辑输出流。I/O 方式变元为 ios::out ©¦ios::binary,它指明流打开仅用于二进制输出。该函数两次调用流函数 write:第 1 次是写数据成员 size,第 2 次是写动态数组的元素。

第 71 到 88 行定义了函数 readArray,它从二进制文件中读取整个数组的元素。参数 filename 指定输入文件的名字。函数打开输入流读出数据成员 size 的值并读取动态数组的元素。如果函数 readArray 从流中成功地读出了数组则返回 true,否则函数将返回 false。该函数用流函数 open 带文件名和 I/O 方式变元打开一逻辑输入流。I/O 方式变元为 ios::in ©¦ios::binary,它指明流打开仅用于二进制输入。该函数两次调用流函数 read:第 1 次是读数据成员 size,第 2 次是读动态数组的值。函数 readArray 的另一特点是它调整了类 Array 的实例的大小以便容纳二进制文件中的数据。这意味着类实例所访问的动态数组可以根据文件中所存储的数组的大小进行缩小或扩大。

清单 12.2 中的成员函数表明该程序进行两类顺序二进制流 I/O 操作。第 1 类 I/O 是在函数 readElem 和 writeElem 中实现的,相关数据项具有同一数据类型。第 2 类 I/O 是在函数 readArray 和 writeArray 中实现的,相关数据项具有不同的数据类型。

函数 main 完成以下操作:

- 第 96 行声明了类 Array 的三个实例,即 arr1, arr2 和 arr3(前两个实例的动态数组大小相同,都由常数 SIZE1 指定;而实例 arr3 更大一些,其大小由常数 SIZE2 指定)。
- 第 97 行声明了文件流 f 并将其打开(用流成员函数)以便用二进制输出方式访问文件 ARR1.DAT。
- 第 100 和 104 行用 for 循环分别为实例 arr1 和 arr3 随意赋值。
- 第 108 行显示实例 arr1 的元素。
- 将数组 arr1 的元素写到输出文件流 f 中,具体是在第 113 行用 for 循环将 writeElem 消息发送给实例 arr1,该消息以输出文件流 f 和循环控制变量 i 为变元。
- 发送 close 消息给输出文件流 f 以关闭输出文件流。
- 在第 118 行打开文件流 f 以访问数据文件 ARR1.DAT(这回,消息 open 指定二进制输入方式)。
- 从输入文件流 f 中读取实例 arr2 的元素(还没有赋值),具体是第 120 行用 for 循环发送消息 readElem 给实例 arr2,该消息的变元为 f、文件流和循环控制变量 i。
- 第 122 行发送 close 消息给输入文件流 f 以关闭输入文件流。

- 第 126 行用 for 循环显示实例 arr2 的元素(这些元素与 arr1 中的相对应)。
- 第 132 行用 for 循环显示实例 arr3 的元素。
- 发送消息 writeArry 给实例 arr3 以一次写入整个实例 arr3(writeArray 消息的文件名变元为 ARR3.DAT)。
- 将数组从文件 ARR3.DAT 中读到实例 arr1 中,发送消息 readArray 给实例 arr1,该消息的变元为文件名 ARR3.DAT。
- 第 143 行用 for 循环显示实例 arr1 中的新的元素。

12.5　随机存取文件流 I/O

随机存取文件流操作也使用上节提到的 read 和 write 函数。流库中提供了大量的流定位函数,用它们可以将流指针移到任何有效位置。函数 seekg 就是其中之一。

语　法

seekg 函数
重载函数 seekg 的原型为:

```
istream& seekg(long pos);
istream& seekg(long pos, seek_dir dir)
```

第 1 个版本中的参数 pos 指定流中的绝对字节位置;第 2 个版本中,参数 pos 所指定的偏移量是相对于参数 dir 取值的。以下是后一个参数的取值:

ios::beg 相对于文件头
ios::cur 相对于文件中的当前位置
ios::end 相对于文件尾

示　例:

```
const BLOCK_SIZE = 80
char buff[BLOCK_SIZE] = "Hello World!";
fstream f("CALC.DAT", ios::in ©¦ios::out ©¦ios::binary);
f.seekg(3 * BLOCK_SIZE); // seek block # 4
f.read((const unsigned char* )buff, BLOCK_SIZE);
cout << buff << "\n";
f.close();
```

新　词

虚数组(virtual array)
虚数组是将定长串存于磁盘上的基于磁盘的数组。

我们来看一个使用随机存取文件流 I/O 的例子。清单 12.3 中列出了程序 IO3.CPP 的源代码,并实现了虚数组。该程序完成以下任务:

- 用内部名字表建立虚数组对象;
- 显示未排序的虚数组对象中的元素;
- 提示输入一字符并按回车键;
- 将虚数组对象的元素排序(这一过程需要随机访问 I/O);

· 显示排序后的虚数组对象的元素。

键　入　　清单 12.3　程序 IO3.CPP 的源代码

```
1: /*
2: C++ program that demonstrates random-access binary file I/O
3: */
4:
5: #include <iostream.h>
6: #include <fstream.h>
7: #include <stdlib.h>
8: #include <string.h>
9:
10: const unsigned MIN_SIZE = 5;
11: const unsigned STR_SIZE = 31;
12: const double BAD_VALUE = -1.0e+30;
13: const boolean { false, true }
14:
15: class VmArray
16: {
17: protected:
18: fstream f;
19: unsigned size;
20: double badIndex
21:
22: public:
23: VmArray(unsigned Size, const char* filename);
24: ~VmArray()
25: { f.close(); }
26: unsigned getSize() const
27: { return size; }
28: boolean writeElem(const char* str, unsigned index);
29: boolean readElem(const char* str, unsigned index);
30: void Combsort();
31: };
32:
33: VmArray::VmArray(unsigned Size, const char* filename)
34: {
35:    char s[STR_SIZE+1];
36:    size = (Size > MIN_SIZE) ? MIN_SIZE : Size;
37:    badIndex = BAD_VALUE;
38:    f.open(filename, ios::in ©¦ios::out ©¦ios::binary);
39:    if (f.good()) {
40: // fill the file stream with empty strings
41: strcpy(s, " ");
42: f.seekg(0);
43: for (unsigned i = 0; i < size; i++)
44   f.write((unsigned char*)s, sizeof(s));
45:       }
46: }
```

```
47:
48: boolean VmArray::writeElem(const char* str, unsigned index)
49: {
50: if (index < size) {
51:     f.seekg(index * (STR-SIZE+1));
52:     f.write((unsigned char*)str, (STR-SIZE+1));
53:     return (f.good()) ? true : false;
54: }
55:     else
56:       return false;
57: }
58:
59: boolean VmArray::readElem(char* str, unsigned index)
60: {
61:     if (index < size)
62:       f.seekg(index * (STR-SIZE+1));
63:       f.read((unsigned char*)str, (STR-SIZE+1));
64:       return (f.good()) ? true : false;
65: }
66:     else
67:     return false;
68: }
69:
70: void VmArray::Combsort()
71: {
72:     unsigned i, j, gap = size;
73:     boolean inOrder;
74:     char strI[STR-SIZE+1], strJ[STR-SIZE+1];
75:
76:     do {
77: gap = (gap * 8) / 11;
78: if (gap < 1)
79:   gap = 1;
80: inOrder = true;
81: for (i = 0, j = gap; i < (size - gap); i++, j++) {
82:   readElem(strI, i);
83:   readElem(strJ, j);
84:   if (strcmp(strI, strJ) > 0) {
85: inOrder = false;
86: writeElem(strI, j);
87: writeElem(strJ, i);
88: }
89: }
90: } while (!(inOrder && gap == 1));
91: }
92:
93: main()
94: {
95:     char* data[] = {"Michigan", "California", "Virginia", "Maine",
96:     "New York", "Florida", "Nevada", "Alaska",
97:     "Ohio", "Maryland" };
98:     VmArray arr(10, "arr.dat");
```

```
99:     char str[STR_SIZE+ 1];
100:    char c;
101:
102:    // assign values to array arr
103:    for (unsigned i = 0; i < arr.getSize(); i++ ) {
104: strcpy(str, data[i]);
105: arr.writeElem(str, i);
106:    }
107:    // display unordered array
108:    cout << "Unsorted arrays is: \n";
109:    for (i = 0; i < arr.getSize(); i++ ) {
110: arr.readElem(str, i);
111: cout << str << "\n";
112:    }
113:    // pause
114:    cout << "\nPress any key and then Return to sort the array...";
115:    cin >> c;
116:    // sort the array
117:    arr.Combsort();
118:    // display sorted array
119:    cout << "Sorted arrays is:\n";
120:    for (i = 0; i < arr.getSize(); i++ ){
121: arr.readElem(str,i);
122: cout << str << "\n";
123:    }
124:    return 0;
125: }
```

以下是清单 12.3 中程序的一个运行范例:

输　出

```
Unsorted arrays is:
Michigan
California
Virginia
Maine
New York
Florida
Nevada
Alaska
Ohio
Maryland
Press any key and then Return to sort the array...
Sorted arrays is:
Alaska
California
Florida
Maine
Maryland
```

```
Michigan
Nevada
New York
Ohio
Virginia
```

分　析

清单 12.3 中的程序声明了类 VmArray。此类建立基于磁盘的数组的模型，该数组将所有的元素存在随机存取的二进制文件中。注意，类中声明了类 fstream 的一个实例，但没有指向动态数组的指针。类中声明了一个构造函数、一析构函数以及一些成员函数。

类构造函数有两个参数，即 Size 和 filename。参数 Size 指定虚数组的大小，参数 filename 指定存储类实例的元素的二进制文件的名字。构造函数用流函数 open，其参数取值是文件名为 filename、I/O 方式为表达式 ios::in ©¦ios::out©¦ios::binary，该表达式指定流以二进制输入和输出方式打开(即随机存取方式)。如果构造函数成功地打开了文件流，它以零填充该文件。类的析构函数完成的动作很简单：关闭文件流 f。

函数 writeElem 和 readElem 支持对数组元素的随机存取。这两个函数用流函数 seekg 将流指针定位于适当的数组元素上，然后 writeElem 调用流函数 write 存储一个数组元素(由参数 str 提供)。与之相对应，函数 readElem 调用流函数 read 检索数组元素(由参数 str 返回)。这两个函数都返回布尔值以指示 I/O 操作是否成功。

类 VmArray 中还声明了函数 Combsort 以对虚数组的元素排序。此函数用 readElem 和 writeElem 成员函数来访问和交换数组元素。

函数 main 完成以下动作：

- 声明类 VmArray 的实例 arr(该实例将 10 个串存于二进制文件 ARR.DAT 中)。
- 为实例 arr 的元素随机赋值，具体是在第 103 到 106 行用 for 循环将 data[i]所访问的字符串赋值给变量 str，然后通过发送消息 writeElem 给实例 arr 将 str 中的值写入 arr(消息 writeElem 的参数取值为串变量 str 和循环控制变量 i)。
- 第 109 行用 for 循环显示未排序的实例 arr 的元素(第 110 行的语句发送消息 readElem 给实例 arr 以获得虚数组的一个元素)。
- 发送消息 Combsort 给实例 arr 以对数组排序。
- 第 120 行用 for 循环显示排序后的实例 arr 的元素(第 121 行中的语句发送消息 readElem 给实例 arr 以获得虚数组中的一个元素)。

小　　结

今天这一课简要地介绍了 C++ 流 I/O 库并讨论了以下内容：

- 通用流函数，包括 open，close，fail 和运算符!。函数 open 打开文件以进行流 I/O 并支持多种 I/O 方式。函数 close 关闭文件流。函数 good 和 fail 分别指示流 I/O 操作的成功与失败。

· 在 C+ + 中,可以用操作符< < 和> > 以及流函数 getline 进行顺序文本流 I/O。操作符< < 可以写字符和字符串(以及其他预定义的数据类型);操作符> > 可用于获得字符。函数 getline 使得应用程序可以从键盘或文本文件中读取字符串。
· 顺序二进制数据流 I/O 使用流函数 write 和 read 读写各种变量的数据。
· 随机存取二进制数据流 I/O 使用 seekg 函数以及 read 和 write 函数。用 seekg 函数可以将流指针移到流中的绝对或相对的字节位置上。

问　与　答

问:如何模拟随机地存取文本文件中的各行?

答:首先以文本方式读出文件中的行,取得行长(每行结尾加上两个字符),并将累计长度存在一名叫 lineIndex 的特殊数组中。该数组存放每行开始的字节位置。最后一个数组元素应保存文件的大小。要想访问第 i 行,可用 seek 或 seekg 函数以定位 lineIndex[i]中的偏移值。第 i 行的长度等于 lineIndex[i+ 1] -lineIndex[i]。

问:如何编写一通用例程以在输入和输出文件流之间进行复制?

答:这需要用流函数 gcount()获得上次的非格式化流输入中实际读取的字节数。以下是函数 copyStream:

```
void copyStream(fstream& fin, fstream& fout,
                unsigned char* buffer, int buffSize)
{
  int n;
  while (fin.read(buffer, buffSize) {
    n = fin.gcount();
    fout.write(buffer, n)
  }
}
```

作　　业

作业中所提供的测验题帮助你巩固所学内容,而练习则就使用所学内容提供一些经验。要弄懂测验和练习的解答后再继续下一天的课程。附录 B 的"答案"中提供了解答。

测　验

1. 对还是错?
 流 I/O 函数 read 和 write 可以正确读写任何数据类型。
2. 对还是错?
 流 I/O 函数 read 和 write 可以正确读写任何数据类型,只要其中没有指针成员。
3. 对还是错?
 如果给 seek 和 seekg 函数提供的索引超出当前文件的末尾一个字节,那么这两个函数会扩展文件。

4. 对还是错?

函数 seek 和 seekg 的变元不需要进行值域检查。

练　习

修改程序 IO3.CPP,建立新程序 IO4.CPP。在 IO4.CPP 中,类 VmArray 应具有函数 binSearch,它对已排序的数组的成员进行二分搜索。在函数 main 末尾加一循环以搜索数组 arr,所用数据为初始化表中的无序数据(此表的成员由指针数据进行存取)。

第13天

用MFC库编写Windows程序

Microsoft 基本类(MFC)库提供了辅助开发 Windows 应用程序的强有力的工具。如果没有这样的库,编写 Windows 应用程序就非常困难、令人沮丧且编写出的代码越来越密集。MFC 库成功地将面向对象和事件驱动的程序设计概念接合起来,并显示了这两种编程思想如何协调工作。今天这一课介绍 MFC 库和一些与 Windows 有关的基本内容,包括用 MFC 库建立 Windows 应用程序。今天要学习以下专题:

- 常用的 Windows 数据类型
- MFC 数据类型约定
- MFC 层次结构
- Windows API 函数
- 响应 Windows 消息
- 发送消息
- 用户定义消息

虽然 MFC 库提供了开发 Windows 应用程序的良好工具,但你还是有必要了解一下底层的 Windows API 函数。

13.1 常用的 Windows 数据类型

Windows 程序设计中用到大量的数据类型。这一节重点介绍与现阶段关系最密切的数据类型。熟悉这些数据类型可以更好地理解 MFC 的声明。Windows 数据类型包括一些简单的数据类型和一组结构。简单数据类型中许多是由 typedef 声明的更具字面含义的别名。这些别名使函数和变量的声明更清晰。表 13.1 中列出了部分最常用的 Windows 数据类型。

表 13.1 最常用的 Windows 数据类型

数据类型	含义(“位”即 bit)
char	8 位有符号字符。
int	16 位有符号整数。
long	32 位有符号整数。
short	16 位有符号整数。
void	空值。
BOOL	表示布尔值的 16 位整数。
BYTE	8 位无符号整数。

续表

数据类型	含义(“位”即 bit)
UINT	16 位无符号整数。
DWORD	32 位无符号整数或段/偏移地址。
FAR	用来建立数据类型的修饰符。
FARPROC	指向函数的长指针。
HANDLE	通用句柄。
HDC	显示描述表的句柄。
HWND	窗口句柄。
LONG	与 long 类型相同。
LPSTR	指向字符串的长指针。
LPVOID	指向 void 类型的长指针。
PWORD	指向 WORD 类型的指针。
WORD	无符号的 16 位整数。

除了表 13.1 中所列的数据类型外,Windows 程序设计中还用到许多结构。POINT 和 RECT 就是非常简单但又很重要的结构。POINT 结构存储一个点的 X 和 Y 坐标,声明如下:

```
typedef struct tagPOINT {
    int x;
    int y;
} POINT;
```

RECT 结构定义矩形的左上角和右下角的坐标,其声明如下:

```
typedef struct tagRECT {
    int left;
    int top;
    int right;
    int bottom;
} RECT;
```

成员 left 和 right 分别代表矩形的左上角和右下角的 X 坐标。top 和 bottom 域则分别代表矩形的左上角和右下角的 Y 坐标。

MFC 库使用匈牙利前缀命名约定,如表 13.2 所示。表中还用此命名约定列举了示范变量。

表 13.2　匈牙利前缀命名约定

前缀	类型	举例	注释
ch	char	chInputChar	8 位字符
b	BOOL	bOk	布尔值
n	int	nCount	整数,长度依系统定
w	WORD	wSize	16 位无符号整数
l	LONG	lTotalSize	32 位有符号整数
dw	DWORD	dwSumSize	32 位无符号整数

续表

前缀	类型	举例	注释
p	*	pStr	指针
lp	FAR*	lpStr	远指针
lpsz	LPSTR	lpszFilename	指向字符串的 32 位指针
lpcz	LPSTR	lpczFilename	指向常量字符串的 32 位指针
h	handle	hWnd	窗口对象的句柄
lpfn	callback	lpfnHelp	指向回调函数的远指针

13.2　MFC 层次结构

MFC 层次结构包含对 Windows 应用程序不同方面进行管理的各个子层次。这些子层次包括 Windows 类、GDI 类、杂用 Windows 类、对象链接和嵌入(OLE)类、文件类、对象 I/O 类、异常类、集合类以及杂项支持类。

CObject 类

CObject 类是 MFC 库的抽象基类,它是 MFC 库和程序员所创建的子孙类的根类。CObject 并不是一个作为摆设的抽象类,相反,它提供了可以通用于所有子孙类的大量操作。这些操作包括对象建立和删除、支持串行化、诊断以及与集合类的相容性。

CMenu 类

CMenu 类是 CObject 类的子孙类,它用来管理菜单。菜单管理包括与窗口有关的菜单资源的建立、更新、跟踪和删除。

文档体系结构(Document Architecture)子层

文档体系结构子层的根类是 CCmdTarget。这一组类包括 CWinApp 类,它是用户定义的应用程序的父类。此子层的另一组类支持 OLE 文档。

窗口框架(Frames Windows)子层

窗口框架子层的根类是 CFrameWnd。这一组类建立模型的基础是基本的单文档界面(SDI)窗口或更复杂的多文档界面(MDI)窗口。Windows 程序管理器就是 MDI 应用程序的一例。后一种窗口使用 CMDIFrameWnd 和 CMDIChildWnd 建立 MDI 应用程序中的框架和子窗口的模型。

控制栏(Control Bars)子层

控制栏子层对工具栏、状态栏、对话栏和分裂窗口建立模型。(Windows 的文件管理器用后者显示目录树和当前选中的目录中的文件清单。)

视窗(Views)子层

视窗子层的根类是 CView。视窗是在其他窗口中创建的子窗口,它可提供用于接受输入的特殊框架。

对话框(Dialog Boxes)子层

对话框子层中包括通用的对话框类 CDialog,还有一些支持常用的对话框的类,可以用这些对话框选择文件、选择颜色、打印以及查找和替换文本。CDialog 类建立模式和非模式对话框模型,它是本子层的根。第 20 天中将讲述 CDialog 类。

控制(Control)子层

控制子层中所包含的类的模型基础是静态文本、命令按钮、位图按钮、列表框、组合框、滚动条、编辑控制和 VB(Visual Basic)控制。这些可视控制所提供的是 Windows 界面的重要组成部分。第 16 到 19 天将重点讲述其中的部分控制。

绘图(Graphical Drawing)子层

CObject 的子孙类 CDC 是 MFC 库中 CDC 子层次的根类。该类及其子孙类支持设备描述对象。CDC 类中包含许多有用的成员函数。这些函数提供的工具有映射函数、绘图工具函数、区域函数——当然这些只是其中的一部分。CDC 类中的成员函数支持由 CDC 的子孙类提供的功能。因而,这些子孙类基本属包容类。

新　词

区域(region)
区域是在窗口内的图形输出所用到的椭圆形或多边形空间。

绘图对象(Graphical Drawing Objects)子层

绘图对象子层建模原型是绘图对象,如笔、刷子、字体、位图和调色板。如果想编写绘制图形对象的应用程序,就需要用到本子层中的类。

其他 MFC 子层

MFC 库还包含其他一些子层,本书中未作论述:

1. 异常(Exceptions)子层中提供的类可以处理各种运行错。
2. 文件服务(File Services)子层提供的类将文件作为对象处理。
3. 集合(Collections)子层中的类可以管理动态对象列表和数组。
4. OLE 1.0 支持(OLE 1.0 Support)子层提供的类可支持对象链接和嵌入。
5. 运行时对象模型支持(Run- Time Object Model Support)子层可提供运行时对象及其基类的有关信息。
6. 简单值类型(Simple Value Types)子层建模原型是一些常用对象,如串、时间、矩形、点和窗口尺寸。
7. 结构(Structures)子层建模原型是与文件状态、打印、内存状态和窗口创建有关的

一些内容。

8. 支持类(Support Classes)子层提供的类可以用来将数据送进送出对话框、管理用户界面产生的命令。

MFC 宏

MFC 库还声明了一些宏以支持 CObject 及其子孙类

- RUNTIME- CLASS 宏返回与指定类相匹配的 CRunTimeClass 结构。
- DECLARE- DYNAMIC 宏可用来访问运行时的类数据。此宏可用在任一类声明中。
- IMPLEMENT- DYNAMIC 宏可用来访问运行时的类数据。此宏用在类实现中。
- DECLARE- SERIAL 宏可以将运行时的类数据串行化并可访问它。
- IMPLEMENT- SERIAL 宏可将运行时的类数据串行化并可访问它。此宏用在类实现中。

13.3　Windows API 函数

如果你熟悉 Windows API 函数则可以跳过这一节,否则接着往下读。这一节是为那些新学编写 Windows 应用程序的读者写的。Windows 3.1 中有 1000 个以上的 Windows API 函数,分以下三组:

- Windows 管理程序接口函数
- 图形设备接口(GDI)函数
- 系统服务接口函数

表 13.3 中列出了 Windows 管理程序接口函数的类型:

新　词

Windows 管理程序接口函数(Windows manager interface function)
Windows 管理程序接口函数管理消息处理、建立窗口、移动窗口、改换窗口以及建立系统输出。

表 13.3　Windows 管理程序接口函数的类型

API 函数类型	用　途
消息	读取和处理应用程序队列中的 Windows 消息。
窗口创建	建立、撤消、更改以及检索有关窗口的信息。
显示和移动	显示、隐藏、移动、返回关于屏幕上窗口的数量和位置的信息。
输入	接管对系统设备的控制,禁止来自系统设备的输入并指定当应用程序接收到来自系统设备的输入时 Windows 所应采取的动作。
硬件	更改及查询输入设备(鼠标和键盘)的状态。
绘画	准备好绘画窗口并提供实际的通用图形函数。
对话框	建立、改变、测试以及删除框内的控制。
滚动	控制窗口及窗口的滚动条的滚动。
菜单	建立、更改以及删除菜单。

续表

API 函数类型	用　途
信息	检索有关屏幕上的窗口数量和位置的信息。
系统	返回有关系统的尺寸、色彩和时间的信息。
剪贴板	在 Windows 应用程序和剪贴板之间进行数据交换。
出错	显示系统错误并请求用户响应以处理错误。
鼠标指示符	创建、删除、显示、隐藏鼠标指示符以及改变其闪烁时间。
光标	设置、移动、显示、隐藏光标以及限制其移动。
挂钩	管理系统的挂钩(Hook)。
特性	建立和查询窗口的特性表。
矩形	设置和查询窗口的客户区中的矩形的有关信息。

新　词

图形设备接口(GDI)函数(graphics device interface function)
图形设备接口函数为 Windows 应用程序提供不依赖于设备图形操作。

图形设备接口(GDI)操作包括往一些输出设备上绘制各种线条、文字和位图映像。表 13.4 中列出了 GDI 函数的类型。MFC 层次结构中还包含对这些函数操作的实现。

表 13.4　图形设备接口(GDI)函数

API 函数类型	用　途
设备描述表	建立、删除和恢复与 Windows 应用程序、设备驱动程序和输出设备相关联的设备描述表。
绘图工具	建立和删除绘图工具,GDI 产生设备输出时要用到这些工具。
调色板	提供访问显示设备色彩的与设备无关的方法。
绘图属性	影响 Windows 应用程序的输出映像(可以是刷子、线条、位图或文本)。
映射	设置和查询 GDI 映射方式信息。
坐标	在用户定义坐标和屏幕坐标之间进行转换,并确定点的位置。
区域	创建、修改和获取有关区域的信息。
剪裁	创建、修改以及测试剪裁区。
线条输出	用选中的画笔工具生成简单的及复杂的线条输出。
椭圆和多边形	画椭圆和多边形。
位图	显示位图。
文本	在设备或显示表面写文本、获取文本信息、改变文本排列及其对齐方式。
字体	选择、创建、删除和获取有关字体的信息。
图元	文件建立、复制、关闭、删除、检索、显示和获取图元文件的有关信息。
设备控制	获取某设备的信息并改变其初始化状态。
打印机控制	获取打印机的性能、改变其初始化状态。
打印机换码	允许应用程序访问特定设备中不能直接通过 GDI 访问到的设施。
环境	设置和查询与输出设备的环境相关的信息。

新　词

系统服务接口函数(system service interface function)
系统服务接口函数访问模块中的代码和数据、分配局部和全局内存、管理任务、装载程序资源、操作串、修改 Windows 初始化文件、提供调试帮助、进行端口通信和文件 I/O 并生成声音。回想一下,一个任务就是一

个 Windows 应用程序调用。

表 13.5 中列出了系统服务接口函数的类型。

表 13.5 系统服务接口函数的类型

API 函数类型	用 途
模块管理	设置和查询 Windows 模块的有关信息。
内存管理	管理全局和局部系统内存。
段	分配、解除分配、锁定、解锁内存段。
OS 中断	允许汇编语言程序无需直接编写中断代码而调用一些 DOS 和 NETBIOS 中断。
任务	改变任务的执行状态、检索与任务有关的信息、以及获取任务执行于其中的环境的信息。
资源管理	从 Windows 应用程序中找出并装入应用资源(仅限于光标、图标、位图、串和字体)。
串操作	提供常用的串操作函数,如串复制、比较、大小写转换和字符测试。
原子管理	生成和管理原子(串的唯一整数索引)。
初始化文件	设置和查询 Windows 初始化文件 WIN.INI 或任何其他私用的初始化文件(通常带.INI 扩展名)中的信息。
通信	通过系统端口进行串行和并行通信。
声音	为系统的发声器产生声音和音乐。
实用宏	获取字节、字以及长整数的内容。
文件 I/O	建立、打开、关闭、以及读写文件。
调试	辅助跟踪 Windows 应用程序及模块中的错误。
优化工具	管理 Windows Profiler 和 Swap 软件开发工具(Microsoft Windows SDK 中有)如何与开发中的 Windows 应用程序进行交互对话。
应用程序执行	允许一个 Windows 应用程序执行另一 Windows 应用程序。

13.4 调用 Windows API 函数

MFC 层次结构的价值在于它用 C+ + 类将大多数 Windows API 函数包装在一起。本节以两个 Windows API 函数为例子讲述如何直接调用这些函数。

以下零散代码段包括了 Polyline, MessageBeep 和 MessageBox API 函数。这些代码的主要目的是画一个多边形,包括以下操作:

- 启动一个试图绘制多边形的循环;
- 调用用户定义的例程以获取用来建立多边形的若干个点;
- 调用布尔型函数 Polyline;
- 如果函数 Polyline 的结果为 FALSE,则调用 API 函数 MessageBeep 以发出“嘟”声,然后再调用 MessageBox 函数。

MessageBox 函数弹出一出错消息对话框,询问是否想得到一组新的绘制点。消息框显示 Yes 和 No 按钮以及小写的 i 图标。

这些步骤的代码如下:

```
const MAX_POINTS = 10;
POINT points[MAX_POINTS];
HDC hDC;
int answer = IDNO;
// other declarations
// other statements
do {
    read_points(points); // get the data for the points
    if (! Polyline(hDC, points, MAX_POINTS)) {
        MessageBeep(0); // beep
        // show error message box
        answer = MessageBox(
                    HWindow,          // window handle
                    "Error! Try again", // error message
                    "Polygon Plot Error!", // message box title
                    MB_YESNO ©¦       // options
                    MB_ICONINFORMATION);
    }
} while (answer = = IDYES)
    // other statements
```

Polyline 函数声明如下:

```
BOOL Polyline(HDC hDC, LPPOINT lpPoints, int nCount)
```

其中, hDC 是设备描述表的句柄。lpPoints 是指向 POINT 型数组的指针。nCount 是数组中用来绘制多边形的点的个数(至少要两个)。

MessageBeep 函数较简单, 其声明如下:

```
void MessageBeep(WORD wType)
```

wType 是唯一的参数, 它取的变元值为 0。

MessageBox 函数是非常重要的 API 函数, 用它可以和你的 Windows 应用程序的最终用户进行通信。MessageBox 函数声明如下:

```
int FAR PASCAL MessageBox(HWND hWndParent, LPSTR lpText,
                          LPSTR lpCaption, WORD wType)
```

第 1 个参数是父窗口的句柄。第 2 和第 3 个参数是消息和标题串。最后一个参数指定出现在消息框中的按钮和图标。表 13.6 中列出了消息框 wType 参数的大部分取值。在上一个例子中, wType 的取值为

```
MB_YESNO ©¦MBICONINFORMATION
```

它让消息框显示 Yes 和 No 按钮并包含小写 i 图标。

表 13.6　消息框 wType 参数的主要取值

取　值	含　义
AB- ABORTRETRYIGNORE	消息框显示三个按钮: Abort，Retry，Ignore。
MB- DEFBUTTON1	第 1 个按钮为默认的。如果未指定 MB- DEFBUTTONx 标识符，则消息框将第 1 个按钮作为默认的。
MB- DEFBUTTON2	第 2 个按钮为默认的。
MB- DEFBUTTON3	第 3 个按钮为默认的。
MB- ICONASTERISK	消息框中包含一图标，它由小写字母 i 外加一圆圈构成，与 MB- ICONINFORMATION 相同。
MB- ICONEXCLAMATION	消息框中包括惊叹号图标。
MB- ICONHAND	消息框中包含一停止符图标，与 MB- ICONSTOP 相同。
MB- ICONINFORMATION	消息框中包含一圆圈内带小写字母 i 组成的图标。
MB- ICONQUESTION	消息框中包含一问号图标。
MB- ICONSTOP	消息框中包含一停止符图标。
MB- OK	消息框中包含 OK 按钮。
MB- OKCANCEL	消息框中包含 OK 和 Cancel 按钮。
MB- RETRYCANCEL	消息框中包含 Retry 和 Cancel 按钮。
MB- YESNO	消息框中包含 Yes 和 No 按钮。
MB- YESNOCANCEL	消息框中包含 Yes，No 和 Cancel 按钮。

MessageBox 函数返回与用户选择相对应的值。表 13.6 列出表示函数值的标识符。

类 CWnd 声明了 MessageBox 成员函数，它是 MessageBox API 函数的包容函数。这两个函数的区别在于函数 CWnd: MessageBox 不带 hWndParent 参数。函数 CWnd::MessageBox 的参数 wType 也使用表 13.6 中所列的值。函数 CWnd::MessageBox 的结果还可以与表 13.7 中所列的值进行比较。

表 13.7　表示 MessageBox 函数的返回值的标识符

标识符	含　义
IDABORT	Abort 按钮被按下。
IDCANCEL	Cancel 按钮被按下。
IDIGNORE	Ignore 按钮被按下。
IDNO	No 按钮被按下。
IDOK	Ok 按钮被按下。
IDRETRY	Retry 按钮被按下。
IDYES	Yes 按钮被按下。

13.5　Windows 消 息

Windows 应用程序中包含各种类型的对象，它们通过消息对事件进行响应，彼此进行交互对话。Windows 就好比是一个办公室，里面有职员、经理、部门、以及资源(计算机、打字机、影印机、电话、传真等等)。每个职员(在我们的讨论中，对应于一个 Windows 对象)的职责是按预定的工作要求(对应于类声明)进行工作。办公室活动的激发因素包括来

自或作用于外部世界的事件以及内部事件。为响应这些事件，各个职员和部门需要通过消息进行交流。类似地，Windows 环境及其应用程序用消息进行交互对话并和外部世界(输入和输出设备)交互作用。这些消息可以用以下方式产生：

- 用户产生的事件，如键盘输入、移动鼠标、按击鼠标按钮等将导致用户产生的消息；
- Windows 应用程序可调用 Windows 函数并导致 Windows 将消息回送给应用程序；
- Winodws 应用程序可以发送内部消息给特定的程序模块；
- Window 环境可以发送消息给 Windows 应用程序；
- 两个 Windows 应用程序可以发送动态数据交换消息给共享数据。

Windows 3.1 中含以下各类消息：

- Windows 管理消息
- 初始化消息
- 输入消息
- 系统消息
- 剪贴板消息
- 系统信息消息
- 控制操作消息
- 控制通知消息
- 滚动条通知消息
- 非客户区消息
- MDI 消息

当 Windows 状态发生改变时，Windows 给应用程序发送 Windows 管理消息。表 13.8 中列出了部分 Windows 管理消息：

表 13.8　部分 Windows 管理消息

消　息	含　义
WM_ACTIVATE	当窗口变为活动或不活动时发送。
WM_CLOSE	关闭窗口时发送。
WM_MOVE	移动窗口时发送。
WM_PAINT	当 Windows 或应用程序请求重画应用程序窗口的一部分时发送。
WM_QUIT	表示终止应用程序的请求。
WM_SIZE	改变窗口大小后发送。

当应用程序构造一个菜单或对话框时，由 Windows 发送初始化消息。表 13.9 中列出了初始化消息。

表 13.9　初始化消息

消　息	含　义
WM_INITDIALOG	显示对话框前发送。
WM_INITMENU	请求初始化菜单。
WM_INITMENUPOPUP	显示弹出式菜单之前发送。

Windows 发送输入消息以响应来自鼠标、键盘、滚动条、或系统计时器的输入。表 13.10 中列出了部分输入消息

表 13.10　部分输入消息

消　息	含　义
WM_COMMAND	选中菜单项后发送
WM_HSCROLL	当用鼠标单击水平滚动条后发送
WM_KEYDOWN	当非系统键被按下后发送
WM_KEYUP	当非系统键被松开后发送
WM_LBUTTONDBLCLK	用鼠标左按钮双击后发送
WM_LBUTTONDOWN	按下鼠标左按钮后发送
WM_LBUTTONUP	松开鼠标左按钮后发送
WM_MOUSEMOVE	移动鼠标时发送
WM_RBUTTONDBLCLK	用鼠标右按钮双击后发送
WM_RBUTTONDOWN	按下鼠标右按钮后发送
WM_RBUTTONUP	松开鼠标右按钮后发送
WM_TIMER	当为特定计时器设置的时限到来时发送
WM_VSCROLL	当用鼠标单击垂直滚动条后发送

当访问 Windows 的控制菜单、滚动条或大小调整框时，Windows 向应用程序发送系统消息。大多数 Windows 应用程序并不对这些消息作出响应，而是将它们传递给 DefWindowProc 函数以进行默认的处理。

当其他应用程序试图访问某窗口的剪贴板时，Windows 向应用程序发送剪贴板消息。

如果在系统一级发生了影响其他 Windows 应用程序的变动，则 Windows 发送系统信息消息。变动因素有：字体、调色板、系统色彩、时间以及 WIN.INI 文件的内容。

控制操作消息由 Windows 应用程序发送给控制对象，如按钮、列表框、组合框以及编辑控制。控制消息可引发执行特定的任务并返回一个表示结果的值。

控制通知消息通知父窗口该窗口所属的控制内所发生的动作。

滚动条通知消息包括 WM_HSCROLL 和 WM_VSCROLL 消息。当单击滚动条时滚动条向其父窗口发送这些消息。

Windows 发送非客户区消息以创建和更新非客户区。在用户自己的 MFC 应用程序中，一般不需要重新设置对这些消息的缺省响应。

新　词

非客户区(nonclient area)
非客户区是窗口的工作区或客户区的外围区域。

MDI 消息由 MDI 窗口框架发送给客户子窗口。这些消息引发的动作包括激活、取消激活、创建、删除、排列以及恢复客户窗口。

13.6　对消息进行响应

MFC 库用一特殊机制确定应用程序如何对不同消息作出响应。这种机制包括一些宏和一个语言扩展功能。C+ + 的扩展功能允许你声明(或标记)消息响应成员函数,这只要将关键字 afx_msg 放在函数的返回类型之前即可。至少含有一个消息响应函数的类须声明宏 DECLARE_MESSAGE_MAP()。该宏初始化程序的实现部分中定义的消息映射。

以下是声明消息响应函数的一个典型例子:

```
class CAppWindow : CFrameWnd {
public:
      CAppWindow();
      // handle painting
      afx_msg void OnPaint();
      // handle resizing a window
      afx_msg void OnSize(UINT nType, int cx, int cy);

protected:
       // handle closing the window
       afx_msg void OnClose();

       DELCARE_MESSAGE_MAP();
};
```

成员函数 OnPaint, OnSize 和 OnClose 分别对预定义消息 WM_PAINT, WM_SIZE 和 WM_CLOSE 进行响应。类实现包括这些成员函数的定义以及特殊的消息映射宏 START_MESSAGE_MAP 和 END_MESSAGE_MAP()。这些特殊宏包含其他一些映射成员函数及其响应的消息的宏。前面的类的消息映射宏为:

```
BEGIN_MESSAGE_MAP(CAppWindow, CFrameWnd)
      ON_WM_PAINT()
      ON_WM_SIZE()
      ON_WM_CLOSE()
END_MESSAGE_MAP()
```

注意 BEGIN_MESSAGE_MAP 有两个变元:应用程序窗口类及其父类 MFC。所用到的类都是必须的而且在多个类具有多个消息响应函数的情况下就更重要。仔细考察三个 ON_WM_xxxx 宏。第 1 个宏 ON_WM_PAINT()指定相应消息 WM_PAINT 的缺省成员函数 OnPaint。实际上,MFC 库中有一大套预定义消息映射宏将各种类成员函数与 WM_xxxx 消息相关联。另有两个例子是 ON_WM_SIZE()和 ON_WM_CLOSE()宏。ON_WM_SIZE()指定响应 WM_SIZE 消息的 OnSize 成员函数。类似地,ON_WM_

CLOSE()宏将 OnClose 成员函数与 WM- CLOSE 消息相对应。

> **注　意**　上面的消息映射声明中消息宏的排列次序与相应的成员函数的声明次序相同，而 Microsoft C+ + 编译器并不要求这样。

13.7　发 送 消 息

在 Windows 应用程序中可以向应用程序本身、其他应用程序或 Windows 系统发送消息。Windows API 函数 SendMessage，PostMessage 和 SendDlgItemMessage 提供了可用来发送消息的重要工具。SendMessage 函数给窗口发送消息并要求窗口处理发送的消息。SendMessage 的声明如下：

```
DWORD SendMessage(HWND hWnd, WORD wMsg, WORD wParam, LONG lParam)
```

参数 hWnd 是接收消息的窗口的句柄。参数 wMsg 指定要发送的消息。参数 wParam 和 lParam 指明附加的任选信息。可以用 SendMessage 函数与其他窗口和控制进行通信。

PostMessage 与 SendMessage 相似，只是少一些紧迫感——该消息被发送到窗口的消息队列中。以后，目标窗口在合适的时候再处理该消息。布尔型的 PostMessage 函数的声明如下：

```
BOOL PostMessage(HWND hWnd, UINT wMsg,
        WPARAM wParam, LPARAM lParam)
```

参数 hWnd 是接收消息的窗口的句柄。参数 wMsg 指定要发送的消息。参数 wParam 和 lParam 指定附加的任选信息。

函数 SendDlgItemMessage 发送消息给对话框中的特定项。SendDlgItemMessage 函数的声明如下：

```
LRESULT SendDlgItemMessage( HWND hWndDlg, int idDlgItem,
        UINT uMsg, WPARAM wParam, LPARAM lParam)
```

参数 hWndDlg 是包含目标控制的对话框的句柄。参数 idDlgItem 指示接收消息的对话框项的整数标识符。参数 wMsg 指定要发送的消息。参数 wParam 和 lParam 指定附加的任选信息。

类 CWnd 声明了给宿主窗口发送消息的成员函数 SendMessage，PostMessage 和 SendDlgItemMessage。这些成员函数声明如下：

```
LRESULT SendMessage(UINT message, WPARAM wParam = 0,
        LPARAM lParam = 0);

BOOL PostMessage(UINT message, WPARAM wParam = 0,
        LPARAM lParam = 0);

LRESULT SendDlgItemMessage(int nID, UINT message,
        WPARAM wParam = 0, LPARAM lParam = 0);
```

13.8　用户定义消息

MFC 允许用户定义自己的消息。常数 WM_USER 与第 1 条消息的数值(见表 13.11)相关联。还需要声明代表用户定义消息的偏移值的常数。例如,可以用# define 指令定义自己的消息:

```
# define WM_USER1 (WM_USER + 0)
# define WM_USER2 (WM_USER + 1)
# define WM_USER3 (WM_USER + 2)
```

表 13.11 中列出了 Windows 3.1 中的 Windows 消息的取值范围。

表 13.11　Windows 消息的范围

常　数	取　值	消息范围	含　义
WM_USER	0x0400	0x0000- 0x03FF	Windows 消息
		0x0400- 0x7FFF	程序员定义的窗口消息
		0x8000- 0xBFFF	保留供 Windows 使用
		0xC000- 0xFFFF	应用程序使用的串消息

用户定义消息指定由菜单项或键盘输入产生的命令。消息映射宏 ON_COMMAND 用来将用户定义命令与其响应函数相关联。ON_COMMAND 宏的一般格式为:

```
ON_COMMAND(command, command_function)
```

为示意如何运用用户定义消息,请看头文件 USRMSG.H 中定义的消息 CM_CALC 和 CM_STORE:

```
# define CM_CALC (WM_USER + 0)
# define CM_STORE (WM_USER + 1)
```

响应这些消息的窗口类声明如下:

```
# include < afxwin.h>
# include "usrmsg.h"

class CAppWindow : CFrameWnd
{
public:
    CAppWindow();

    // handle painting
    afx_msg void OnPaint();

    // handle calculation command
    afx_msg void CMCalc();

    // handle store command
    afx_msg void CMStore();
```

```
protected:

    // handle closing the window
    afx_msg void OnClose();

    DECLARE_MESSAGE_MAP();
};
```

相应的消息映射为:

```
BEGIN_MESSAGE_MAP(CAppWindow, CFrameWnd)
    ON_WM_PAINT()
    ON_COMMAND(CM_CALC, CMCalc)
    ON_COMMAND(CM_STORE, CMStore)
    ON_WM_CLOSE()
END_MESSAGE_MAP()
```

第 1 个 ON_COMMAND 宏指定响应 CM_CALC 命令消息的成员函数 CMCalc; 第 2 个 ON_COMMAND 宏指定响应 CM_STORE 命令消息的成员函数 CMStore。再次提醒一句, 消息宏的次序与消息响应成员函数的声明次序是一致的。

MFC 库中还有其他一些消息映射宏以在需要时使用。

小　　结

今天这一课讲解了 Microsoft 基本类(MFC)的类层次结构的基本内容以及与 Windows 有关的一些内容。这些内容有:

- 常用 Windows 数据类型包括简单数据类型(如 BYTE, WORD 和 LONG)和数据结构(如 RECT 和 POINT);
- MFC 层次结构中包含组成 MFC 库的各种类。
- 有三种通用的 Windows API 函数: 窗口管理函数、GDI 函数和系统服务函数;
- 可以调用的 Windows API 函数包括 MessageBox, MessageBeep 和 Polyline;
- 在用户自己的 MFC 应用程序中可以对几类 Windows 消息进行响应。这一过程中要声明 MFC 子孙类, 其中包含一个或多个消息响应函数, 这些函数可提供所需的响应;
- 可以用一些 Windows API 函数发送消息;
- 可以定义自己的消息、发送消息并对它们进行响应。

第 14 天要讲解建立 MFC 应用程序以及为这类应用程序配置菜单的基本知识。

问　与　答

问: MFC 是由各类 Windows API 函数的简单包容函数所组成的吗?

答: MFC 2.0 版中具有比 1.0 版更复杂的类。新类中有更多的内置函数。

问: 匈牙利标记法在 Visual C+ + 中是否是必须的?

答: 不, 这只是个命名约定, 目的是为了使你的 C+ + 代码更清晰。采用任何命名标记都需要一定的时间, 因此不用操之过急。

问: Windows 发送给实例的消息与 C+ + 发送给实例的消息之间有什么联系?

答: 两者都使用了对象作为类比物。

作　　业

作业中所提供的测验题帮助你巩固所学内容, 而练习则就使用所学内容提供一些经验。要弄懂测验和练习的解答后再继续下一天的课程。附录 B 的“答案”中提供了解答。

测　验

1. 如果要创建的消息框中包含 Yes, No 和 Cancel 按钮以及问号图标, 那么需要用哪一个 MB_xxxx 常数?
2. 如果要创建的消息框中包含 OK 按钮以及惊叹号图标, 那么需要用哪一个 MB_xxxx 常数?
3. 如果在一包含了 Yes 和 No 按钮的消息对话框中单击 Yes 按钮, 那么 MessageBox 函数返回什么值?
4. 如果在一包含了 Yes, No 和 Cancel 按钮的消息对话框中单击 Cancel 按钮, 那么 MessageBox 函数返回什么值?

练　习

用 Visual C+ + 包中的快速参考(Quick Reference)卡熟悉 MFC 层次结构。

第14天

建立基本的MFC应用程序

第13天我们讲述了MFC基本类库、Windows API函数和Windows消息的有关背景内容。今天这一课开始学习简单的MFC应用程序的建立。你将学习以下专题:

- 建立小的MFC应用程序
- 对应用程序窗口操作的扩展
- 为MFC应用程序增加菜单
- 对菜单选择进行响应
- 在不同菜单之间进行切换
- 建立MFC应用程序的多个实例
- 关闭窗口

今天这一课还将展现MFC库的威力和功能,其功能包括缺省的特性和响应。建立自己的MFC应用程序时要扩展MFC的基本特性和功能。

14.1 建立一个小的MFC应用程序

每次建立MFC应用程序时,一开始都要声明CWinApp的子孙类。即使是最简单的MFC应用程序,这一步也是必需的。对MFC库类的其他扩展依应用程序而定。Windows应用程序的入口处是WinMain函数(它类似于非Windows的C或C++程序中的main()函数)。幸运的是,MFC库中提供了适合多数Windows应用程序的默认的WinMain函数。也可以定义自己的WinMain版本以便在更高层次上编写应用程序。本书中的应用程序使用预定义的WinMain函数。若要执行一MFC程序,可声明一应用程序实例,如以下的一般格式中所示:

```
- CWindowApp- - WindowApp- ;
```

类- CWindowApp- 代表从CWinApp派生出的应用程序类。变量- WindowApp- 代表应用程序实例。此实例由类构造函数或继承类的构造函数生成。

当MFC应用程序开始运行时,它按以下顺序调用成员函数:

1. 调用虚成员函数InitApplication初始化MFC应用程序的第1个实例。InitApplication成员函数的代码如下:

```
BOOL CWinApp::InitApplication()
{ return TRUE; }
```

此成员函数不做任何动作——它只是一个空函数,你可以重写它以对应用程序

进行特定的初始化。

2. 调用虚成员函数 InitInstance 以初始化 MFC 应用程序的其他各实例。CWinApp::Instance 成员函数与 InitApplication 类似，也是一个空函数。每创建 MFC 应用程序的一个实例都要调用这两个函数。
3. 调用窗口类构造函数初始化每个实例的主窗口。

注　意　如果在初始化应用程序实例时还想执行另一些步骤，则需要按以下的一般格式声明自己的 InitInstance:

```
BOOL_ CWinApp_ ::InitInstance()
{
   // put statements here
   return boolean_ value;
}
```

语　法

应用程序类

应用程序类的语法一般为:

```
class CWindowApp_ :  public CWinApp
{
public:
      // public data members declarations
protected:
// protected data members

// optional constructor
CWindowApp_ (const char* pszAppName,      // application name
             _ other_ params_ );          // additional parameter

// other constructors
// class destructors
// other member functions

// overridable data members
virtual BOOL InitInstance();
virtual BOOL InitApplication(); // optional

// other protected member functions

private:
      // private data members
      // other private member functions
};
```

这一通用模板表明用户的应用程序类应声明成员函数 InitInstance。而成员函数 InitApplication 仅在想要微调 MFC 应用程序中的这些继承函数时才需要声明。这一点也适用于其他构造函数参数、数据成员(公有、保护和私有)以及成员函数(公有、保护和私有)。

示　例:

见清单 14. 1。

我们来看一个例子。清单 14. 1 **中列出了程序** MINWINAP. CPP **的源代码。此程序是一个很小的** Windows **应用程序。可以用鼠标或菜单系统中可选菜单项来对程序窗口进行移动、重置大小、使最小化、使最大化等操作。若要关闭窗口, 可使用系统菜单中的** Close **项。程序除了这些操作以外什么也不做。例如, 如果用户用鼠标单击, 什么也不会发生。**

> **注　意**　**要建立** Windows **应用程序还需要一个**. DEF **文件。清单** 14. 2 **中列出了文件** MINWINAP. DEF **的内容。此文件的内容包括应用程序名、标题、操作方式以及堆大小。本书中的所有**. DEF **文件都与** MINWINAP. DEF **非常相似, 只是应用程序名和标题不一样。**

为了生成程序 MINWINAP. EXE, **先在** MSVC\ VC21DAY **目录下建立子目录** MINWINAP。**将文件** MINWINAP. CPP **和** MINWINAP. DEF **存在** MSVC\ VC21DAY\ MINWINAP **目录下, 该目录还用于保存由** Visual C+ + **工作平台生成的** MINWINAP. MAK **文件。**

键　入　　**清单** 14. 1　**文件** MINWINAP. CPP **中的源代码**

```
1: # include < afxwin. h>
2:
3: // Define a window class derived from CFrameWnd
4: Class CAppWindow：  public CFrameWnd
5: {
6: public:
7:      CAppWindow()
8:      { Create(NULL, "Minimal MFC Application",
9:                      WS_OVERLAPPEDWINDOW, rectDefault); }
10: };
11:
12: // Define an application class derived from CWinApp
13: Class CWindowApp：  public CWinApp
14: {
15: public:
16: virtual BOOL InitInstance();
17: };
18:
19: // Construct the CWindowApp's m_pMainWnd data member
20: BOOL CWindowApp::InitInstance
21: {
22:   m_pMainWnd =  new CAppWindow();
23:   m_pMainWnd-  > ShowWindow(m_nCmdShow);
24:   m_pMainWnd-  > UpdateWindow();
```

```
25:    return TRUE;
26: }
27:
28: // application's constructor initializes and runs the app
29: CWindowApp WindowApp;
```

键　入　　清单 14.2　MINWINAP.DEF 文件的内容

```
1: NAME          MinWinAp
2: DESCRIPTION   'Minimal MFC Windows Application'
3: EXETYPE       WINDOWS
4: CODE          PRELOAD MOVEABLE DISCARDABLE
5: DATA          PRELOAD MOVEABLE MULTIPLE
6: HEAPSIZE      1024
```

图 14.1 中显示了程序 MINWINAP.EXE 的一个运行范例。

图 14.1　程序 MINWINAP.EXE 的一个运行范例

分　析

清单 14.1 中列出了一个小的 MFC 应用程序的源代码，该程序中使用了今天这一课前面所提到的模板。程序中声明了应用程序类 CWindowApp(CWinApp 的子孙类)以及窗口类 CAppWindow(CWnd 的子孙类)。

第 1 行中的指令将头文件 AFXWIN.H 包含进来。该文件可确保所有必需的 Windows 和 MFC 声明都可提供给编译器。

第 13 行中声明的类 CWindowApp 中仅包含成员函数 InitInstance。该函数建立 CAppWindow 的实例,由继承的指针类型的数据成员 m_pMainWnd 访问。

第 4 行声明了窗口类 CAppWindow。这个类中仅声明了类构造函数,构造函数用成员函数 CFrameWnd::Create 建立类实例。CFrameWnd::Create 成员函数声明如下:

```
BOOL Create(
    const char FAR* lpClassName,                    // Windows class
                                                    // registration name
    const char FAR* lpWindowName,                   // title
    DWORD dwStyle = WS_OVERLAPPEDWINDOW,            // style attribute
    const RECT& rect = rectDefault,                 // location and size
    const CWnd* pParentWnd = NULL,                  // parent window
    const char FAR* lpMenuName = NULL);             // ID of menu
```

lpClassName 是 Windows 注册类的名字。这一非 OOP 名字指定窗口类别名。参数 lpWindowName 指定窗口的标题或名称。参数 dwStyle 指定所生成的窗口的风格,其默认值为 WS_OVERLAPPEDWINDOW。参数 rect 定义窗口的位置和尺寸,其默认值为 rectDefault。参数 pParentWnd 是指向父窗口的指针,默认值为 NULL。参数 lpMenuName 定义了窗口所配置的菜单资源的名字。有些菜单资源具有与常数相对应的数值 ID(标识符)。若想使用这类菜单资源,可调用宏 MAKEINTRESOURCE 并提供所要菜单的数值 ID。该宏将数值 ID 转换成串。

新　词

资源(resource)

资源是在 Windows 应用程序外建立并编译的程序成分。

第 29 行语句声明了应用程序实例。这一声明将引发执行 MFC 程序。

14.2　扩充窗口操作

建立小的 MFC 应用程序之后,我们在该应用程序中增加一些操作,编写出一个新的应用程序。下面的程序完成以下操作:

- 通过显示带 OK 按钮的消息框响应鼠标左按钮的单击动作(当鼠标在应用程序窗口内时);
- 通过显示询问是否想关闭窗口的消息框响应鼠标右按钮的单击动作;
- 支持包含嵌套菜单选择项(除 Exit 以外的菜单项都为空)的菜单。

大多数 Windows 应用程序都包含菜单。MINWINAP.EXE 应用程序只有系统菜单。这一节说明如何用资源文件建立为应用程序窗口配置的菜单。本例在创建窗口时从菜单资源中装入菜单,下一节再讲述更复杂的菜单操作。

应用程序要使用资源时就将其装入。资源仅定义可视对象的外观,它们不对这些对象的行为负责,因为那是 Windows 或 MFC 应用程序的任务。使用资源的优越性在于无需重新编译应用程序就可修改它们。例如,可以修改菜单项上文字或改变按钮的位置,更具

戏剧性的是还可以修改不同的菜单、对话框和其他控制中所用的语言。这样，当对每一种语言都使用不同版本的资源时，软件开发者就可以很容易地维护和更新一个应用程序版本。Apple 计算机公司就采用这种方法使 Apple Macintosh 计算机能工作于不同的语言环境，包括阿拉伯语。

> **注　意**　附录 A 中叙述了菜单和对话框的资源描述语法。其第 1 节讨论菜单资源。

清单 14.3 中列出了资源文件 MENU1.RC 的内容。文件中定义了一个菜单，其标识名字是 OPTIONS。菜单有三个主项：File，Edit 和 Help。File 和 Edit 项是弹出式菜单。Edit 菜单中的 Delete 项也是弹出式菜单。菜单项中还包括水平分隔条。所有菜单名中都用“&”符号定义相应的热键。

各 MENUITEM 语句中所包含的 CM_xxxx 常数定义选中菜单后的结果。常数 CM_xxxx 在头文件 MENU1.H 中定义，如清单 14.4 所示。清单 14.5 中列出了程序 MENU1.CPP 的源代码。

在目录\MSVC\VC21DAY 下建立子目录 MENU2 并将所有项目文件存在该目录中。该项目的.MAK 文件中包括了文件 MENU1.CPP，MENU1.RC 和 MENU1.DEF。后一个文件与清单 14.2 中所列出的很相似。

键　入　　**清单 14.3　资源描述文件 MENU1.RC**

```
1: # include < windows.h>
2: # include < afxres.h>
3: # include "menus.h"
4:
5: OPTIONS MENU LOADONCALL MOVEABLE PURE DISCARDABLE
6: BEGIN
7:   POPUP "&File"
8:   BEGIN
9:     MENUITEM "&New", CM_FILENEW
10:    MENUITEM "&Open", CM_FILEOPEN
11:    MENUITEM "&Save", CM_FILESAVE
12:    MENUITEM "Save&As", CM_FILESAVEAS
13:    MENUITEM SEPARATOR
14:    MENUITEM "E&xit", CM_EXIT
15:  END
16:  POPUP "&Edit"
17:  BEGIN
18:    MENUITEM "&Undo", CM_EDITUNDO
19:    MENUITEM SEPARATOR
20:    MENUITEM "C&ut", CM_EDITCUT
21:    MENUITEM "C&opy", CM_EDITCOPY
22:    MENUITEM "&Paste", CM_EDITPASTE
23:    POPUP "&Delete"
```

```
24:      BEGIN
25:       MENUITEM "&Line", CM_EDITDELETE
26:       MENUITEM "&Block", CM_EDITDELETE_BLOCK
27:      END
28:      MENUITEM "&Clear", CM_EDITCLEAR
29:    END
30:    MENUITEM "&Help", CM_HELP
31:  END
```

键　入　　**清单 14.4　头文件 MENU1.H 的源代码**

```
 1: # define CM_FILENEW              (WM_USER + 100)
2: # define CM_FILEOPEN             (WM_USER + 101)
3: # define CM_FILESAVE             (WM_USER + 102)
4: # define CM_FILESAVEAS           (WM_USER + 103)
5: # define CM_EXIT                 (WM_USER + 104)
6: # define CM_EDITUNDO             (WM_USER + 105)
7: # define CM_EDITCUT              (WM_USER + 106)
8: # define CM_EDITCOPY             (WM_USER + 107)
9: # define CM_EDITPASTE            (WM_USER + 108)
10: # define CM_EDITDELETE          (WM_USER + 109)
11: # define CM_EDITDELETE_BLOCK    (WM_USER + 110)
12: # define CM_EDITCLEAR           (WM_USER + 111)
13: # define CM_HELP                (WM_USER + 112)
```

键　入　　**清单 14.5　程序 MENU1.CPP 的源代码**

```
1: # include < afxwin.h>
2: # include "menu1.h"
3:
4: // Define a window class derived from CFrameWnd
5: Class CAppWindow：  public CFrameWnd
6: {
7: public:
8: CAppWindow()
9:   { Create(NULL, "A Menu- Driven MFC Application",
10:            WS_OVERLAPPEDWINDOW, rectDefault, NULL, "OPTIONS"); }
11:
12: protected:
13:   // handle the left mouse button click
14:   afx_msg void OnLButtonDown(UINT nFlags, CPoint point);
15:   // handle the right mouse button click
16:   afx_msg void OnRButtonDown(UINT nFlags, CPoint point);
17:   // handle the Exit menu option
18:   afx_msg void OnExit();
```

```
19:
20:   // message map macro
21:   DELCARE- MESSAGE- MAP()
22: };
23:
24: // Define an application class derived from CWinApp
25: Class CWindowApp：  public CWinApp
26: {
27: public:
28:   virtual BOOL InitInstance();
29: }
30:
31: void CAppWindow::OnLButtonDown(UINT nFlags, CPoint point)
32: {
33:   MessageBox("You clicked the left button!",
34:              "Mouse Click Event!", MB- OK)
35: }
36:
37: VOID CAppWindow::OnRButtonDown(UINT nFlags, CPoint point)
38: {
39:   MessageBeep(0); // beep
40:   // prompt user if he or she want to close the application
41:   if (MessageBox("Want to close this application", "Query",
42:                  MB- YESNO ©¦MB- ICONQUESTION) = =  IDYES)
43:   SendMessage(WM- CLOSE);
44: }
45:
46: void CAppWindow::OnExit()
47: { SendMessage(WM- CLOSE); }
48:
49: BEGIN- MESSAGE- MAP(CAppWindow, CFrameWnd)
50:       ON- WM- LBUTTONDOWN()
51:       ON- WM- RBUTTONDOWN()
52:       ON- COMMAND(CM- EXIT, OnExit)
53: END- MESSAGE- MAP()
54:
55: // Construct the CWindowApp's m- pMainWnd data member
56: BOOL CWindowApp::InitInstance
57: {
58:   m- pMainWnd =  new CAppWindow();
59:   m- pMainWnd- > ShowWindow(m- nCmdShow);
60:   m- pMainWnd- > UpdateWindow();
61:   return TRUE;
62: }
63:
64: // application's constructor initializes and runs the app
65: CWindowApp WindowApp;
```

图 14.2 中显示了程序 MENU1.EXE 的一个运行范例。

图 14.2　程序 MENU1.EXE 的一个运行范例

分　析

清单 14.5 中的程序声明了 CFrameWnd 的子孙类以实施所选动作。第 5 行中声明了 CFrameWnd 的子孙类 CAppWindow, 它声明了一个构造函数、三个消息响应成员函数和宏 DECLARE_MESSAGE_MAP。构造函数 CAppWindow 的工作方式与程序 MINWINAP.CPP 中的类似。注意第 9 到 10 行对 Create 函数的调用语句所带的变元表以串 OPTIONS 结尾, 它指定了要装入的菜单资源名。菜单资源名与清单 14.3 的第 5 行中的一致。成员函数 OnLButtonDown 和 OnRButtonDown 分别响应 Windows 消息 WM_LBUTTONDOWN 和 WM_RBUTTONDOWN。BEGIN_MESSAGE_MAP 宏也对此响应作出回应, 它指定由各个成员函数处理的消息。类 CAppWindow 还声明了成员函数 OnExit 以响应 Exit 菜单选项。

第 31 到 35 行定义了成员函数 OnLButtonDown。此函数仅用来调用继承的 CWnd::MessageBox 成员函数以在标题为“Mouse Click Event!”消息框中显示串“You clicked the left button!”。该消息框中还有一 OK 按钮, 单击它可恢复程序执行。

第 46 到 47 行定义了成员函数 OnExit。此函数对 Exit 菜单选项所产生的 CM_EXIT 命令进行响应。此命令与 Windows 消息命令 WM_COMMAND 相对应。

第 37 到 44 行定义了成员函数 OnRButtonDown。此函数先调用 Windows API 函数 MessageBeep 以发出“嘟”声, 然后再调用函数 CWnd::MessageBox 以显示标题为 Query 的方框、消息“Want to close this application”、Yes 和 No 按钮以及问号图标。第 41 行的 if 语句将函数 MessageBox 的结果与预定义常量 IDYES 进行比较。如果这两个值一致, 函数用 WM_CLOSE 消息调用成员函数 CWnd::SendMessage 以关闭窗口。

第 49 到 53 行定义了宏 BEGIN_MESSAGE_MAP 和 END_MESSAGE_MAP。这些宏包括消息映射宏 ON_WM_LBUTTONDOWN，ON_WM_RBUTTONDOWN 和 ON_COMMAND。前两个宏不带变元，因此它们使用缺省的映射。接着，成员函数 OnLButton 和 OnRButton 分别对 WM_LBUTTONDOWN 和 WM_RBUTTONDOWN 消息作出响应。宏 ON_COMMAND 截取 CM_EXIT 菜单命令并调用 OnExit 成员函数处理它。

> **注　意**　Microsoft 在推广 Visual C++ 及其工作平台时特别强调 AppWizard，ClassWizard 和 App Studio 工具，有了它们，开发 Windows 应用程序和建立资源就很容易。这些专家工具对于使建立基于 MFC 的应用程序的重复性的工作自动化是非常有效的。它们可生成各种源文件以得到复杂的 Windows 应用程序。但是，如果你是个初学者，则还需要学习 MFC 应用程序的基本组成才能使用 App Wizard 和 ClassWizard。一旦学会了如何用手工编写基于 MFC 的 Windows 程序，就可以使用 AppWizard 了，因为你能够区分主要代码和从属代码。

14.3　响应菜单选择

上一个程序演示了如何生成嵌套菜单以及在应用程序中如何装入它。但是，选中菜单项(除 Exit 外)不产生任何动作。应用程序中如何响应菜单选择，在响应时是否还能装入不同的菜单？这一节主要回答这一问题。

我们来看看清单 14.6 中列出的资源文件 MENU2.RC 的内容。资源文件中定义了两个菜单，分别标记为 LONGMENU 和 SHORTMENU。正如名字所表示的，第 1 个菜单是第 2 个的长版本。这些菜单与 MENU1.RC 中的不同，它们不是嵌套的且使用了 INACTIVE 和 GRAYED 选项。注意这两个选项并不是在每个菜单中都出现——它们被赋给那些确实存在的但非活动的菜单项。结果，这些菜单项的名字都以灰色出现，不能被选中。从 C++ 编码方面来说，没有相应的消息响应成员函数。相反，那些不带 INACTIVE 和 GRAYED 选项的菜单项可以被选中且会产生响应，虽然是简单地响应。菜单资源中所用的常数 CM_xxxx 都是在 MENU2.H 头文件中定义的，如清单 14.7 所示。

清单 14.6 的菜单项包括两个有趣的成员。菜单 LONGMENU 中的 Short &Menus 项将产生选择短菜单的 CM_SHORTMENU 命令。菜单 SHORTMENU 有一补充菜单项 &Long Menu，它产生选择长菜单的 CM_LONGMENU 命令。

在\MSVC\VC21DAY 下建立子目录 MENU2 并将所有项目文件存于新目录下。该项目的.MAK 文件中应包含 MENU2.CPP，MENU2.RC 和 MENU2.DEF。后一个文件与清单 14.2 中的 MINWINAP.DEF 相似。运行应用程序时，可用 Long Menu 和 Short Menu 菜单选项在两种形式的菜单之间进行切换。另外，选中 Help 项可以得到联机帮助消息框。选中 Edit Cut，Edit Copy 以及 Edit Paste 项，观察反应。在不同菜单之间移动时

注意非活动菜单项。清单 14.8 中的程序包括头文件 MENU2.H。这种包含使得用户可以在消息响应成员函数中使用自己定义的 CM_xxxx 常量。源代码中声明了应用程序类 CWindowApp 和窗口类 CAppWindow。第 2 个类负责响应各 Windows 消息。类 CAppWindow 声明了构造函数、析构函数以及一组消息响应成员函数。类的构造函数创建窗口并生成实例 CMenu。该实例由类 CAppWindow 的数据成员 pAppMenu 指针访问。构造函数然后装入 LONGMENU 资源，将菜单对象与窗口相关联并调用成员函数 DrawMenuBar。接着，应用程序的窗口以长菜单方式启动。

键　入　　清单 14.6　资源描述文件 MENU2.RC

```
1: #include <windows.h>
2: #include <afxres.h>
3: #include "menu2.h"
4:
5: LONGMENU MENU LOADONCALL MOVEABLE PURE DISCARDABLE
6: BEGIN
7:   POPUP "&File"
8:   BEGIN
9:     MENUITEM "&New", CM_FILENEW, INACTIVE, GRAYED
10:    MENUITEM "&Open", CM_FILEOPEN, INACTIVE, GRAYED
11:    MENUITEM "&Save", CM_FILESAVE, INACTIVE, GRAYED
12:    MENUITEM "Save&As", CM_FILESAVEAS, INACTIVE, GRAYED
13:    MENUITEM SEPARATOR
14:    MENUITEM "Short &Menus", CM_SHORTMENU
15:    MENUITEM SEPARATOR
16:    MENUITEM "E&xit", CM_EXIT
17:  END
18:  POPUP "Edit"
19:  BEGIN
20:    MENUITEM "&Undo", CM_EDITUNDO, INACTIVE, GRAYED
21:    MENUITEM SEPARATOR
22:    MENUITEM "C&ut", CM_EDITCUT
23:    MENUITEM "C&opy", CM_EDITCOPY
24:    MENUITEM "&Paste", CM_EDITPASTE
25:    MENUITEM "&Delete", CM_EDITDELETE, INACTIVE, GRAYED
26:    MENUITEM "&Clear", CM_EDITCLEAR, INACTIVE, GRAYED
27:  END
28:  MENUITEM "&Help", CM_HELP
29: END
30:
31: SHORTMENU MENU LOADONCALL MOVEABLE PURE DISCARDABLE
32: BEGIN
33:  POPUP "&File"
34:  BEGIN
35:    MENUITEM "&Open", CM_FILEOPEN, INACTIVE, GRAYED
36:    MENUITEM "Save&As", CM_FILESAVEAS, INACTIVE, GRAYED
37:    MENUITEM SEPARATOR
38:    MENUITEM "&Long Menus", CM_LONGMENU
```

```
39:      MENUITEM SEPARATOR
40:      MENUITEM "E&xit", CM_EXIT
41:    END
42:    POPUP "&Edit"
43:    BEGIN
44:      MENUITEM "C&ut", CM_EDITCUT
45:      MENUITEM "C&opy", CM_EDITCOPY
46:      MENUITEM "&Paste", CM_EDITPASTE
47:    END
48:    MENUITEM "&Help", CM_HELP
49: END
```

键　入　　**清单 14.7　头文件 MENU2.H 的内容**

```
1: #define CM_FILENEW              (WM_USER + 100)
2: #define CM_FILEOPEN             (WM_USER + 101)
3: #define CM_FILESAVE             (WM_USER + 102)
4: #define CM_FILESAVEAS           (WM_USER + 103)
5: #define CM_EXIT                 (WM_USER + 104)
6: #define CM_EDITUNDO             (WM_USER + 105)
7: #define CM_EDITCUT              (WM_USER + 106)
8: #define CM_EDITCOPY             (WM_USER + 107)
9: #define CM_EDITPASTE            (WM_USER + 108)
10: #define CM_EDITDELETE          (WM_USER + 109)
11: #define CM_EDITDELETE_BLOCK    (WM_USER + 110)
12: #define CM_EDITCLEAR           (WM_USER + 111)
13: #define CM_HELP                (WM_USER + 112)
14: #define CM_SHORTMENU           (WM_USER + 113)
15: #define CM_LONGMENU            (WM_USER + 114)
```

键　入　　**清单 14.8　文件 MENU2.CPP 的源代码**

```
1: #include <afxwin.h>
2: #include "menu2.h"
3:
4: // Define a window class derived from CFrameWnd
5: Class CAppWindow : public CFrameWnd
6: {
7: public:
8:      CMenu*  pAppMenu;
9:      CAppWindow();
10:      ~CAppWindow() { if(pAppMenu) delete pAppMenu; }
11: protected:
12:      // handle the left mouse button click
13:      afx_msg void OnLButtonDown(UINT nFlags, CPoint point);
```

```
14:     // handle the right mouse button click
15:     afx_msg void OnRButtonDown(UINT nFlags, CPoint point)
16:     // load the long menu
17:     afx_msg void OnLongMenu();
18:     // load the short menu
19:     afx_msg void OnShortMenu();
20:     // exit the application
21:     afx_msg void OnExit();
22:     // get on-line help
23:     afx_msg void OnHelp();
24:     // handle the Cut menu option
25:     afx_msg void OnEditCut();
26:     // handle the Copy menu option
27:     afx_msg void OnEditCopy();
28:     // handle the Paste menu option
29:     afx_msg void OnEditPaste();
30: private:
31:     // display a message "Feature not implemented"
32:     void notImplemented();
33:     // prompt user to quit the application
34:     void promptToQuit();
35:     // Assign new menu
36:     void assignMenu(const char FAR* lpMenuName);
37:     // message map macro
38:     DECLARE_MESSAGE_MAP()
39: };
40:
41: // Define an application class derived from CWinApp
42: Class CWindowApp : public CWinApp
43: {
44: public:
45:   virtual BOOL InitInstance();
46: };
47:
48: CAppWindow::CAppWindow()
49: {
50:   Create(NULL, "Multi-Menu Application",
51:             WS_OVERLAPPEDWINDOW, rectDefault);
52:   pAppMenu = new CMenu;
53:   pAppMenu->LoadMenu("LONGMENU");
54:   SetMenu(pAppMenu);
55:   DrawMenuBar();
56: }
57:
58: void CAppWindow::OnLButtonDown(UINT nFlags, CPoint point)
59: {
60:   MessageBox("You clicked the left button!",
61:               "Mouse Click Event!", MB_OK);
62: }
63:
64: void CAppWindow::OnRButtonDown(UINT nFlags, CPoint point)
65: { promptToQuit(); }
```

```
66:
67: void CAppWindow::OnLongMenu()
68: { assignMenu("LONGMENU"); }
69:
70: void CAppWindow::OnShortMenu()
71: { assignMenu("SHORTMENU"); }
72:
73: void CAppWindow::OnExit()
74: { SendMessage(WM_CLOSE); }
75:
76: void CAppWindow::OnHelp()
77: {
78:   MessageBox("This is a sample one line help",
79:    "Help", MB_OK ¦ MB_ICONINFORMATION);
80: }
81:
82: void CAppWindow::OnEditCut()
83: { notImplemented(); }
84:
85: void CAppWindow::OnEditCopy()
86: { notImplemented(); }
87:
88: void CAppWindow::OnEditPaste()
89: { notImplemented(); }
90:
91: void CAppWindow::notImplemented()
92: {
93:   MessageBox("This feature is not implemented",
94:              "Information", MB_OK);
95: }
96: void CAppWindow::promptToQuit()
97: {
98:   MessageBeep(0); // beep
99:   // prompt user if he or she want to close the application
100:   if (MessageBox("Want to close this application", "Query",
101:                  MB_YESNO ¦ MB_ICONQUESTION) == IDYES)
102:      SendMessage(WM_CLOSE);
103: }
104:
105: void CAppWindow::assignMenu(const char FAR* lpMenuName)
106: {
107:   SetMenu(NULL);
108:   pAppMenu->DestroyMenu();
109:   pAppMenu->LoadMenu(lpMenuName);
110:   SetMenu(pAppMenu);
111:   DrawMenuBar();
112: }
113:
114: BEGIN_MESSAGE_MAP(CAppWindow, CFrameWnd)
115:   ON_WM_LBUTTONDOWN()
116:   ON_WM_RBUTTONDOWN()
117:   ON_COMMAND(CM_LONGMENU, OnLongMenu)
```

```
118:    ON_COMMAND(CM_SHORTMENU, OnShortMenu)
119:    ON_COMAMND(CM_EXIT, OnExit)
120:    ON_COMMAND(CM_HELP, OnHelp)
121:    ON_COMMAND(CM_EDITCUT, OnHelp)
122:    ON_COMMAND(CM_EDITCOPY, OnCopy)
123:    ON_COMMAND(CM_EDITPASTE, OnPaste)
124: END_MESSAGE_MAP()
125:
126: // Construct the CWindowApp's m_pMainWnd data member
127: BOOL CWindowApp::InitInstance()
128: {
129:    m_pMainWnd = new CAppWindow()
130:    m_pMainWnd->ShowWindow(m_nCmdShow);
131:    m_pMainWnd->UpdateWindow();
132:    return TRUE;
133: }
134:
135: // application's constructor initialize and runs the app
136: CWindowApp WindowApp;
```

图 14.3 中显示了程序 MENU2.EXE 的一个运行范例。

图 14.3　程序 MENU2.EXE 的一个运行范例

分　析

类 CAppWindow 声明了三个私有函数: notImplemented, promptToQuit 和 assignMenu。第 1 个函数显示了包含消息“This feature is not implemented”的消息框。实

际上,大多数菜单项的反应与此相同。函数 promptToQuit 用带有 Yes 和 No 按钮的消息框提示你关闭应用程序。成员函数 assignMenu 删除当前菜单并装入新菜单。程序用此函数在短菜单和长菜单之间切换。

现在我们来看 CAppWindows 的消息响应成员函数。前两个函数与上一个应用程序中的相同——对鼠标在窗口中的按击作出响应。新版本的 OnRbuttonDown 函数仅调用函数 promptToQuit。

成员函数 OnLongMenu 对用户定义的命令 CM_LONGMENU 作出响应。消息映射宏 ON_COMMAND(CM_LONGMENU, OnLongMenu)指明函数 OnLongMenu 对 CM_LONGMENU 命令消息作出响应。OnLongMenu 以"LONGMENU"为变元调用成员函数 assignMenu 以装入长菜单,成员函数 OnShortMenu 的代码与其类似,它响应 CM_SHORTMENU 命令以装入短菜单。

成员函数 OnExit 提供了对 Exit(它产生 CM_EXIT 命令)菜单选项的新的响应动作。这个函数用继承的函数 SendMessage 发送 Windows 消息 WM_CLOSE。

成员函数 OnHelp 产生一联机帮助消息框以响应用户定义的 CM_HELP 消息。消息框中有一 OK 按钮以及一信息图标(圆圈内包含字母 i)。

成员函数 OnEditCut, OnEditCopy 和 OnEditPaste 都要调用私有成员函数 notImplemented,这是对 CM_EDITCUT, CM_EDITCOPY 和 CM_EDITPASTE 命令的相同的(象征性的)响应。

消息循环宏包括几个 ON_COMMAND 宏,它们将各 CM_xxxx 命令消息与相应的消息响应函数相关联。

14.4　建立多个实例

Windows 的强大功能之一就是它可以运行同一应用程序的多个实例。支持这一特性的是一种巧妙的设计,它通过共享相同的代码和数据段从而节省了内存和系统资源。因而,你可以装入例如一个科学计算器的三个实例且独立地用它们来工作。这一节讲述建立 MFC 应用程序的多个实例。为使例子程序更有趣味性,第 1 个实例的窗口标题和菜单与其他实例都不相同。所有实例以相同的方式响应鼠标左右按钮的按击。清单 14.9 中列出了资源描述文件 MENU3.RC,其中定义了各个程序实例的菜单资源。文件 MENU3.RC 定义了两个菜单资源:MAINMENU 和 SECONDMENU。这些菜单资源与 MENU2.RC 中的菜单 LONGMENU 和 SHORTMENU 相似,只是没有切换菜单项。清单 14.10 中列出了头文件 MENU3.H。清单 14.11 中列出了应用程序文件 MENU3.CPP 的源代码。

在\MSVC\VC21DAY 下建立子目录 MENU3 并将所有项目文件存在该目录下。该项目的.MAK 文件中应包含文件 MENU3.CPP, MENU3.RC 和 MENU3.DEF。后一个文件与清单 14.2 中的 MINWINAP.DEF 文件很相似。运行程序 MENU3.EXE 的多个实例。第 1 个实例的窗口标题为 Main Window,而其他实例的标题都为"A Secondary Windows"。此外,第 1 个实例的 File 菜单项中的选项比其他实例中的多。

键　入　　清单 14.9　资源描述文件 MENU3.RC

```
1: # include < windows.h>
2: # include < afxres.h>
3: # include "menu3.h"
4:
5: MAINMENU MENU LOADONCALL MOVEABLE PURE DISCARDABLE
6: BEGIN
7:   POPUP "&File"
8:   BEGIN
9:     MENUITEM "&New", CM_FILENEW, INACTIVE, GRAYED
10:    MENUITEM "&Open", CM_FILEOPEN, INACTIVE, GRAYED
11:    MENUITEM "&Save", CM_FILESAVE, INACTIVE, GRAYED
12:    MENUITEM "Save&As", CM_FILESAVEAS, INACTIVE, GRAYED
13:    MENUITEM SEPARATOR
14:    MENUITEM "E&xit", CM_EXIT
15:  END
16:  POPUP "&Edit"
17:  BEGIN
18:    MENUITEM "&Undo", CM_EDITUNDO, INACTIVE, GRAYED
19:    MENUITEM SEPARATOR
20:    MENUITEM "C&ut", CM_EDITCUT
21:    MENUITEM "C&opy", CM_EDITCOPY
22:    MENUITEM "&Paste", CM_EDITPASTE
23:    MENUITEM "&Delete", CM_EDITDELETE, INACTIVE, GRAYED
24:    MENUITEM "&Clear", CM_EDITCLEAR, INACTIVE, GRAYED
25:  END
26:  MENUITEM "&Help", CM_HELP
27: END
28:
29: SECONDMENU MENU LOADONCALL MOVEABLE PURE DISCARDABLE
30: BEGIN
31:  POPUP "&File"
32:  BEGIN
33:    MENUITEM "&Open", CM_FILEOPEN, INACTIVE, GRAYED
34:    MENUITEM "Save&As", CM_FILESAVEAS, INACTIVE, GRAYED
35:    MENUITEM SEPARATOR
36:    MENUITEM "E&xit", CM_EXIT
37:  END
38:  POPUP "&Edit"
39:  BEGIN
40:    MENUITEM "C&ut", CM_EDITCUT
41:    MENUITEM "C&opy", CM_EDITCOPY
42:    MENUITEM "&Paste", CM_EDITPASTE
43:  END
44:  MENUITEM "&Help", CM_HELP
45: END
```

键　入　　**清单 14.10　头文件 MENU3.H 的源代码**

```
1: # define CM_FILENEW            (WM_USER + 100)
2: # define CM_FILEOPEN           (WM_USER + 101)
3: # define CM_FILESAVE           (WM_USER + 102)
4: # define CM_FILESAVEAS         (WM_USER + 103)
5: # define CM_EXIT               (WM_USER + 104)
6: # define CM_EDITUNDO           (WM_USER + 105)
7: # define CM_EDITCUT            (WM_USER + 106)
8: # define CM_EDITCOPY           (WM_USER + 107)
9: # define CM_EDITPASTE          (WM_USER + 108)
10: # define CM_EDITDELETE        (WM_USER + 109)
11: # define CM_EDITDELETE_BLOCK (WM_USER + 110)
12: # define CM_EDITCLEAR         (WM_USER + 111)
13: # define CM_HELP              (WM_USER + 112)
14: # define CM_SHORTMENU         (WM_USER + 113)
15: # define CM_LONGMENU          (WM_USER + 114)
```

键　入　　**清单 14.11　文件 MENU3.CPP 的源代码**

```
1: # include < afxwin.h>
2: # include "menu3.h"
3:
4: // Define a window class derived from FrameWnd
5: class CAppWindow : public CFrameWnd
6: {
7: public:
8:      CWppWindow(BOOL, IsMainWindow);
9:
10: protected:
11:      // handle the left mouse button click
12:      afx_msg void OnLButton(UINT nFlags, CPoint point);
13:      // handle the right mouse button click
14:      afx_msg void OnRButton(UINT nFlags, CPoint point);
15:      // exit the application
16:      afx_msg void OnExit();
17:      // get on- line help
18:      afx_msg void OnHelp();
19:      // handle the Cut menu option
20:      afx_msg void OnEditCut();
21:      // handle the Copy menu option
22:      afx_msg void OnEditCopy();
23:      // handle the Paste menu option
24:      afx_msg void OnEditPaste();
25:
26: private:
```

```
27:     // display a message "Feature not implemented"
28:     void notImplemented();
29:     // prompt user to quit the application
30:     void promptToQuit();
31:     // message map macro
32:     DECLARE_MESSAGE_MAP()
33: };
34:
35: // Define an application class derived from CWinApp
36: class CWindowApp : public CWinApp
37: {
38: public:
39:   BOOL IsFirst;
40:   virtual BOOL InitInstance();
41:   virtual BOOL InitApplication();
42: };
43:
44: CAppWindow::CAppWindow(BOOL IsMainWindow)
45: {
46:   if (IsMainWindow)
47:    Create(NULL, "Main Window",
48:           WS_OVERLAPPEDWINDOW, rectDefault, NULL, "MAINMENU");
49:   else
50:     Create(NULL, "A Secondary Window",
51:            WS_OVERLAPPEDWINDOW, rectDefault, NULL, "SECONDMENU");
52: }
53:
54: void CAppWindow::OnLButtonDown(UINT nFlags, CPoint point)
55: {
56:   MessageBox("You clicked the left button!",
57:              "Mouse Click Event!", MB_OK);
58: }
59:
60: void CAppWindow::OnRButtonDown(UINT nFlags, CPoint point)
61: { promptToQuit(); }
62:
63: void CAppWindow::OnExit()
64: { SendMessage(WM_CLOSE); }
65:
66: void CAppWindow::OnHelp()
67: {
68:   MessageBox(
69:    "This is a sample one line help",
70:    "Help", MB_OK ©¦MB_ICONINFORMAION);
71: }
72:
73: void CAppWindow::OnEditCut()
74: { notImplemented(); }
75:
76: void CAppWindow::OnEditCopy()
77: { notImplemented(); }
78:
```

```
79: void CAppWindow::OnEditPaste()
80: { notImplemented(); }
81:
82: void CAppWindow::notImplemented()
83: {
84: MessageBox("This feature is not implemented",
85:             "Information", MB_OK);
86: }
87:
88: void CAppWindow::promptToQuit()
89: {
90:   MessageBeep(0); // beep
91:
92:   // prompt user if he or she want to close the application
93:   if (MessageBox("Want to close this application", "Query",
94:                  MB_YESNO ©¦MB_ICONQUESTION) == IDYES)
95:      SendMessage(WM_CLOSE);
96: }
97:
98: BEGIN_MESSAGE_MAP(CAppWindow, CFrameWnd)
99:      ON_WM_LBUTTONDOWN()
100:     ON_WM_RBUTTONDOWN()
101:     ON_COMMAND(CM_EXIT, OnExit)
102:     ON_COMMAND(CM_HELP, OnHelp)
103:     ON_COMMAND(CM_EDITCUT, OnEditCut)
104:     ON_COMMAND(CM_EDITCOPY, OnEditCopy)
105:     ON_COMMAND(CM_EDITPASTE, OnEditPaste)
106: END_MESSAGE_MAP()
107:
108: // Construct the CWindowApp's m_pMainWnd data member
109: BOOL CWindowApp::InitInstance()
110: {
111:   m_pMainWnd = new CAppWindow(IsFirst);
112:   IsFirst = FALSE;
113:   m_pMainWnd->ShowWindow(m_nCmdShow);
114:   m_pMainWnd->UpdateWindow();
115:   return TRUE;
116: }
117:
118: // Initialize the application
119: BOOL CWindowApp::InitApplication()
120: {
121:   IsFirst = TRUE;
122:   return IsFirst;
123: }
124:
125: // application's constructor initializes and runs the app
126: CWindowApp WindowApp;
```

图 14.4 中显示了 MENU3.EXE 应用程序的一个运行范例。

图 14.4　应用程序 MENU3.EXE 的一个运行范例

分　析

清单 14.11 中的程序如何建立与众不同的第 1 个实例?在本例中,第 1 个实例的窗口标题和菜单与其他实例都不相同。为实现这一点,程序在应用程序类 CWindowApp 中声明了布尔型数据成员 IsFirst。类中还声明了成员函数 InitApplication 和 InitInstance。第 119 到 123 行中定义了成员函数 CWindowApp::InitApplication,它给数据成员 IsFirst 赋值 TRUE。这一赋值使应用程序可以确定所建立的实例是否是第 1 个。第 109 到 116 行所定义的成员函数用变元 IsFirst(被初始化为 TRUE)建立一个窗口实例。函数然后给成员函数赋值 FALSE。这个值在后面的实例中并不改变。

程序声明的构造函数 CAppWindow 带有布尔型参数 IsMainWindow。如果参数 IsMainWindow 的取值为 TRUE,构造函数以"Main Window"为标题、MAINMENU 为菜单资源建立一个窗口。否则,构造函数以"Secondary Window"为标题、SECONDMENU 为菜单资源建立一个窗口。

14.5　关 闭 窗 口

回想一下,在程序 MENU1.EXE 到 MENU3.EXE 中,可以通过选择 Exit 菜单项退出应用程序并关闭其窗口。这些应用程序用默认的响应程序快速地终止程序。虽然这种突然退出对前面的程序不会构成危害,但是如果坚持这样做的话,软件开发者会被用户的投诉信所淹没的。因此,Windows 应用程序只有在确信用户不会丢失数据的情况下才能

关闭其窗口。此外,应用程序还应进行其他一些相应的关闭操作。

> **注　意**　Windows 和 MFC 中都提供了一套复杂的步骤以小心地关闭应用程序及其所有子窗口。实际上,这一过程就好像对有关对象进行的查询,这些对象可以阻止为响应 WM_CLOSE 消息而关闭目标窗口。通过在自定义的窗口中声明成员函数 OnClose,在关闭窗口这件事上给应用程序又提供了一次发言权(或表决权)。你所声明的 OnClose 成员函数的焦点在于关闭你的应用程序的特殊判定标准。如果有新的或更改的数据未保存,那么多数情况下应保证用户不会退出应用程序。

我们来看一个合理关闭应用程序窗口的程序。清单 14.12 中列出了程序 MENU4.CPP 的源代码。这个程序是从 MENU1.CPP 中派生出来的。文件 MENU4.H 和 MENU4.RC 分别与 MENU1.H 和 MENU1.RC 文件相同,因而没有列出。

在\MSVC\VC21DAY 下建立子目录 MENU4 并将所有项目文件存在该目录下。该项目的.MAK 文件中应包含文件 MENU4.CPP, MENU4.RC 和 MENU3.DEF。后一个文件与清单 14.2 中的文件 MINWINAP.DEF 相似。运行该程序并选择 Exit 菜单项,程序的反应是显示一个带 Yes 和 No 按钮的消息框,询问你是否愿意关闭应用程序。单击鼠标右按钮时也会得到同样的消息框。因此,程序 MENU4.EXE 强化了对退出应用程序的管理。

键　入　　**清单 14.12　程序文件 MENU4.CPP 的源代码**

```
1: # include < afxwin.h>
2: # include "menu4.h"
3:
4: // Define a window class derived from CFrameWnd
5: class CAppWindow : public CFrameWnd
6: {
7: public:
8:   CAppWindow()
9:     { Create(NULL, "A Menu-Driven MFC Application",
10:      WS_OVERLAPPEDWINDOW, rectDefault, NULL, "OPTIONS"); }
11:
12: protected:
13:   // handle the left mouse button click
14:   afx_msg void OnLButtonDown(UINT nFlags, CPoint point);
15:   // handle the right mouse button click
16:   afx_msg void OnRButtonDown(UINT nFlags, CPoint point);
17:   // handle the Exit menu option
18:   afx_msg void OnExit();
19:   // handle closing the window
20:   afx_msg void OnClose();
21:
22:   // message map macro
```

```
23:   DELCLARE_MESSAGE_MAP()
24: };
25:
26: // Define an application class derived from CWinApp
27: class CWindowApp : public CWinApp
28: {
29: public:
30:   virtual BOOL InitInstance();
31: };
32:
33: void CAppWindow::OnLButtonDown(UINT nFlags, CPoint point)
34: {
35:   MessageBox("You clicked the left button!",
36:              "Mouse Click Event!", MB_OK);
37: }
38:
39: void CAppWindow::OnRButtonDown(UINT nFlags, CPoint point)
40: { SendMessage(WM_CLOSE); }
41:
42: void CAppWindow::OnExit()
43: { SendMessage(WM_CLOSE); }
44:
45: void CAppWindow::OnClose()
46: {
47:   if (MessageBox("Want to close this application",
48:                  "Query", MB_YESNO ¦MB_ICONQUESTION) == IDYES)
49:     DestroyWindow();
50: }
51:
52: BEGIN_MESSAGE_MAP(CAppWindow, CFrameWnd)
53:     ON_WM_LBUTTONDOWN()
54:     ON_WM_RBUTTONDOWN()
55:     ON_COMMAND(CM_EXIT, OnExit)
56:     ON_WM_CLOSE()
57: END_MESSAGE_MAP()
58:
59: // Construct the CWindowApp's m_pMainWnd data member
60: BOOL CWindowApp::InitInstance()
61: {
62:   m_pMainWnd = new CAppWindow();
63:   m_pMainWnd->ShowWindow(m_nCmdShow);
64:   m_pMainWnd->UpdateWindow();
65:   return TRUE;
66: }
67:
68: // application's constructor initializes and runs the app
69: CWindowApp WindowApp;
```

图 14.5 中显示了程序 MENU4.EXE 的一个运行范例。

图 14.5　程序 MENU4.EXE 的一个运行范例

分　析

程序 MENU4.CPP 和 MENU1.CPP 的最重要的区别在自定义的窗口类 CAppWindow 中声明了成员函数 OnClose:

```
void CAppWindow::OnClose()
{
    if (MessageBox("Want to close this application",
                   "Query", MB_YESNO ¦ MB_ICONQUESTION) == IDYES)
        DestroyWindow();
}
```

函数 OnClose 中含一个 if 语句,其中调用了继承的成员函数 MessageBox。消息框显示了询问消息,Yes 和 No 按钮以及信息图标。函数 OnClose 将 MessageBox 的结果与常数 IDYES 进行比较。如果条件为真,函数将调用继承函数 DestroyWindow。

声明 OnClose 成员函数还使你可以重写 OnRButtonDown 成员函数中的语句。新版本中,第 41 到 42 行中的代码很简单,它们取决于函数 OnClose。OnRButton 只是用 SendMessage 成员函数发送 WM_CLOSE 消息,而 OnClose 成员函数负责处理此消息。

小　结

今天这一课讲述了建立 MFC 应用程序的基本知识。其内容如下:

· 可以建立提供预定义操作和默认响应的小型 MFC 应用程序;
· 扩展窗口操作处理鼠标单击从而显示简单的消息框;
· 可以增加具有嵌套菜单结构的菜单以示意如何用资源描述文件建立复杂的菜单;
· 有多种方法可以响应菜单选择,包括在运行时改变菜单结构;
· 可以建立 MFC 应用程序的多个实例并使第 1 个实例具有不同的窗口标题和菜单结构;
· 成员函数 OnClose 对关闭窗口进行确认。

今天这一课开始让你感觉 MFC 如何像组织公司雇员一样组织其应用程序部件,而每一天的程序所提供的简单交互操作仅仅涉及到 Windows 和 MFC 的复杂世界的一小部分。明天的课程将焦点集中在窗口类上并提供一个在窗口中放置文本的应用程序。

问 与 答

问:如果成员函数 Create 试图装入一个并不存在的菜单资源时会发生什么情况?
答:运行系统装入应用程序后将不会有窗口出现。此外,非法的应用程序可能导致 Windows 出运行错。
问:消息处理成员函数为何要声明为保护的?
答:这种声明可以防止函数 CWindow::InitInstance 通过指针 m_pMainWnd 直接调用消息处理函数。
问:预定义的消息响应映射是什么?
答:宏 BEGIN_MESSAGE_MAP 可包括自动调用成员函数 OnXxxx 以处理 WM_XXXX 消息对应的 ON_WM_XXXX 宏。例如,ON_WM_CLOSE 宏映射函数 OnClose 以处理消息 WM_CLOSE。
问:消息映射宏中映射项的排列次序必须与窗口类中相应的成员函数的声明次序相同吗?
答:不,这种次序是任意的。

作 业

作业中所提供的测验题帮助你巩固所学内容,而练习则就使用所学内容提供一些经验。要弄懂测验和练习的解答后再继续下一天的课程。附录 B 的“答案”中提供了解答。

测 验

1. 以下说法是否正确?
 ON_WM_CREATE 消息映射项将函数 OnCreate 与 Windows 消息 WM_CREATE 相关联。
2. 以下说法是否正确?
 ON_WM_ACTIVATE 消息映射项将函数 Onactivate 与 Windows 消息 WM_

CREATE 相关联。

3. 以下说法是否正确?

 ON_COMMAND(CM_CLEAR, OnClear)将函数 OnClear 与 Windows 消息 CM_CLEAR 相关联。

4. 以下说法是否正确?

 如果把程序 MENU4.CPP 的第 40 行改成

```
{ SendMessage(WM_COMMAND, CM_EXIT); }
```

 不会改变程序的交互动作。

练　习

用 App Studio, AppWizard 和 ClassWizard 建立与程序 MENU1.CPP 相类似的程序。按 Microsoft 手册中的指导使用这些专家工具。注意由专家工具所生成的文件的数量、大小以及内容和今天这一课中的有什么差别。

第2周

内容复习

学习用 Visual C+ + 进行程序设计的第 2 周已经过去,我们来看一个猜数游戏的增强版本。清单 R2.1 中列出了程序 GAME2.CPP 的源代码。虽然这一版本与用户交互对话的方式和 GAME1.CPP 相同,但它使用了类和枚举类型。

键　入　　清单 R2.1　程序 GAME2.CPP 的源代码

```
1: # include < stdlib.h>
2: # include < iostream.h>
3: # include < time.h>
4:
5: enum boolean { false, true };
6:
7: // declare a global random number generating function
8: int random(int maxVal)
9: { return rand() % maxVal; }
10:
11: class game
12: {
13:   protected:
14:         int n;
15:         int m;
16:         int MaxIter;
17:         int iter;
18:         boolean ok;
19:         void prompt();
20:         void examineInput();
21:
22: public:
23:    game();
24:    void play();
25: };
26:
27: game::game()
28: {
29:   MaxIter =  11;
30:   iter =  0;
31:   ok =  true;
32:
33:   // reseed random-  number generator
```

```
34:   srand((unsigned)time(NULL));
35:   n = random(1001);
36:   m = - 1;
37: }
38:
39: void game::prompt()
40: {
41:   cout << "Enter a number between 0 and 1000 : ";
42:   cin >> m;
43:   ok = (m < 0) ? false : true;
44: }
45:
46: void game::examineInput()
47: {
48:   // is the user's guess higher?
49:   if (m > n)
50:         cout << "Enter a lower guess\n\n";
51:   else if (m < n)
52:         cout << "Enter a higher guess\n\n";
53:   else
54:         cout << "You guessed it! Congratulations.";
55: }
56:
57: void game::play()
58: {
59:   // loop to obtain the other guesses
60:   while (m != n && iter < MaxIter && ok) {
61:    prompt();
62:    iter++;
63:    examineInput();
64:   }
65:   // did the user guess the secret number
66:   if (iter >= MaxIter ©©¦ok == 0)
67:         cout << "The secret number is "<< n << "\n"
68: }
69:
70: main()
71: {
72:   game g;
73:
74:   g.play();
75:   return 0;
76: }
```

以下是清单 R2.1 中的程序的一个运行范例:

输　出

```
Enter a number between 0 and 1000 : 500
Enter a lower guess
```

```
Enter a number between 0 and 1000 : 250
Enter a higher guess

Enter a number between 0 and 1000 : - 1
Enter a higher guess

The secret number is 324
```

分　析

清单 R2. 1 中的程序声明了枚举类型 boolean 以建立布尔型值的模型。程序中还声明了类 game, 它建立猜数游戏的模型。类中包含一些数据成员, 其中有布尔型变量 ok。此外, 类中还声明了保护成员函数 prompt 和 examineInput, 还有公有的构造函数以及成员函数 play。

构造函数初始化数据成员并给随机数发生器重新‘播种”。第 39 到 44 行中定义的成员函数 prompt 提示你输入、获取输入并根据输入为变量 ok 赋值 true/ false。

第 46 到 55 行中定义的函数 examineInput, 将你的猜测(存在数据成员 m 中)与秘密数字(存在数据成员 n 中)相比较并显示适当的消息。

第 57 到 68 行中定义的数据成员 play 中包含运行游戏的 while 循环。循环语句调用成员函数 prompt 和 examineInput 且增大数据成员 iter 的值。另外, 函数所包含的 if 语句在猜测失败或你想退出游戏时显示秘密数字。

函数 main 声明了类 game 的实例 g 并发送消息 play 给该实例。

第3周

概要

最后一周我们将进一步介绍如何用 MFC 库中的各种类来建立 Windows 应用程序，学习怎样创建滚动窗口，此外，还将学习如何使用静态文本控制、编辑控制、下压按钮、选择框、圆按钮、分组控制、列表框、滚动条、组合框、对话框和 MDI 窗口。这些控制组成了 Windows 应用程序最常用的直观控制。本周的每一天都将利用应用程序来演示这些控制的各种特性。

第15天

基本窗口

Windows 环境中最实质性的内容就是窗口的使用。

新　词

窗口(window)
一般地,窗口是指屏幕上含有特定信息的可视部分。

窗口是显示信息的场所,当改变窗口尺寸、移动窗口或使用现有滚动条时,应用程序负责信息的维护。今天这一课,我们将学习如下内容:

- 建立只读文本窗口
- 利用滚动条滚动文本
- 改变滚动条参数(单位、行大小、页大小和范围)
- 优化 OnPaint 成员函数

15.1　建立只读文本窗口

第 14 天中曾给出了一些使用菜单驱动的 MFC 应用程序,当然,在这些窗口中,并没有显示任何信息。这一节将提供一个在窗口中显示只读文本的 MFC 应用程序,这个程序的基本部分与只读的联机帮助窗口相似。

该程序主要用于演示如何显示文本以及当如下的一种或多种情况发生时怎样对文本进行维护:

- 改变窗口的尺寸
- 使窗口最小化、正常化或使之最大化
- 在文本区移动模式对话框

应用程序的这类特点,主要是通过成员函数 CDC::TextOut 和 CWnd::OnPaint 实现的,函数 TextOut 在指定的显示设备上画出一个字符串,并用当前选定的字体和指定的窗口坐标显示。

语　法

OutText 函数
重载的成员函数 Text Out 声明如下:

```
BOOL TextOut(int X, int Y, char FAR* lpString, int nCount);
BOOL TextOut(int X, int Y, CString& str);
```

参数 X 和 Y 指定第一个字符出现在窗口中的位置,参数 lpString 指向将在窗口中显示的字符串,参数 nCount 为 lpString 的打头的部分中要显示的字符数,它通常取值为要

显示的字符串变元的长度,函数 TextOut 的第二种格式中,参数 str 为 CString 的实例的引用。函数在成功时返回非零值,否则返回零。

示　例:

```
char s[81] = "Hello";
CString cs = "Bon Jour!";
TextOut(10, 20, s, strlen(s));
TextOut(20, 10, cs);
```

> **注　意**　MFC 库中提供了建立字符串对象模型的类 CString。类 CString 在 Visual C+ + 中较特别,建议你去读一下 Visual C+ + 产品包中所带的 MFC 库参考书中有关此类的说明。

可以想见,函数 TextOut 只是一次性地显示文本,这就意味着,当我们改变窗口的可见部分或移动一个窗口,并从显示的文本上通过时,会被抹掉文本。因此需要一种可以刷新窗口中的显示文本的机制。使用成员函数 CWnd::OnPaint 便能够显示和维护窗口(文本以及图形)的内容。当调用函数 UpdateWindow 时,函数 OnPaint 也将被调用,以响应 WM_PAINT 消息。刷新也包括窗口的初始建立——窗口的尺寸从无到有。因此,OnPaint 的多功能包括设置初始显示,并对其进行维护。

> **注　意**　在窗口应用程序类中必须声明一个 OnPaint 成员函数。在 OnPaint 函数中放置的程序代码用来确定信息的显示、保留以及消失。

本 MFC 应用程序中,自始至终显示的都是相同的信息。成员函数 CWnd::OnPaint 的一般形式为:

```
void CAppWindow::OnPaint()
{
  CPaintDC dc(this);
  // other declarations

  // statements using the TextOut member function
  // e.g.
  //        dc.TextOut(x, y, s, strlen(s));
}
```

局部变量 dc 声明了 CPaintDC 的一个实例,并用值 this 初始化,以指向应用程序窗口。这一声明已足够建立设备描述表对象,函数 OnPaint 则可以在需要时调用函数 dc.TextOut。注意,在用 C 编写的使用了 TextOut API 函数的的 Windows 应用程序中,实现相似的文本输出需要初始化设备描述表对象且在完成任务后释放它,但在 MFC 应用程序中,这些步骤将由 CDC 类自动执行。

现在我们来看看 MFC 应用程序的代码,清单 15.1 中列出了 WINDOW1.H 头文件,它只定义了 CM_EXIT。清单 15.2 中包含了资源描述文件 WINDOW1.RC,这个资源文件定义了一个只含一个选项的菜单 Exit 用于退出应用程序。清单 15.3 是 WINDOW1.CPP 程序的源代码。

在目录\MSVC\VC21DAY 下建立一个新的子目录 WINDOW1,把所有的项目文件都放在该子目录下。项目的.MAK 文件应包含文件 WINDOW1.CPP, WINDOW1.RC 和 WINDOW1.DEF,后一文件与第 14 天中清单 14.2 的 MINWINAP.DEF 相似。

键　入　**清单 15.1　头文件 WINDOW1.H 的源代码**

```
1: # define CM_EXIT (WM_USER + 100)
```

键　入　**清单 15.2　资源描述文件 WINDOW1.RC**

```
1: # include < windows.h>
2: # include < afxres.h>
3: # include "window1.h"
4:
5: EXITMENU MENU LOADONCALL MOVEABLE PURE DISCARDABLE
6: BEGIN
7:     MENUITEM "E&xit", CM_EXIT
8: END
```

键　入　**清单 15.3　WINDOW1.CPP 程序文件的源代码**

```
1: # include < afxwin.h>
2: # include < stdio.h>
3: # include < string.h>
4: # include "window1.h"
5:
6: // Define a window class derived from CFrameWnd
7: class CAppWindow : public CFrameWnd
8: {
9: public:
10:    CAppWindow()
11:    { Create(NULL, "A Simple Read- Only Text Window",
12:        WS_OVERLAPPEDWINDOW, rectDefault, NULL, "EXITMENU"):}
13:
14: protected:
15:   // handle the left mouse button click
16:   afx_msg void OnLButtonDown(UINT nFlags, CPoint point);
17:   // handle the Exit menu option
18:   afx_msg void OnExit();
19:   // handle closing the window
20:   afx_msg void OnClose();
21:   // handle painting the window
22:   afx_msg void OnPaint();
```

```
23:
24:    // message map macro
25:    DECLARE_MESSAGE_MAP();
26: };
27:
28: // Define an application class derived from CWinApp
29: class CWindowApp:  public CwinApp
30: {
31: public
32:    virtual BOOL InitInstance();
33:
34: };
35:
36: void CAppWindow::OnLButtonDown(UINT nFlags, CPoint point)
37: {
38:    MessageBox("You clicked the left button!",
39:                "Mouse Click Event!", MB_OK);
40: }
41:
42:
43: void CAppWindow::OnExit()
44: { SendMessage(WM_CLOSE); }
45:
46: void CAppWindow::OnClose()
47: {
48:    if (MessageBox("Want to close this application",
49:                    "Query", MB_YESNO ©¦MB_ICONQUESTION) == IDYES)
50:       DestroyWindow();
51: }
52:
53: void CAppWindow::OnPaint()
54: {
55:    const MAX_LINES = 30;
56:    const LINE_INCR = 20;
57:    char s[81];
58:    CPaintDC dc(this):
59:    BOOL ok = 1;
60:
61:    for (int i = 0; i < MAX_LINES && ok; i++) {
62:       sprintf(s, "This is line number %d", i);
63:       ok = dc.TextOut(0, i* LINE_INCR, s, strlen(s));
64:    }
65: }
66:
67: BEGIN_MESSAGE_MAP(CAppWindow, CFrameWnd)
68:     ON_WM_LBUTTONDOWN()
69:     ON_WM_RBUTTONDOWN()
70:     ON_COMMAND(CM_EXIT, OnExit)
71:     ON_WM_CLOSE()
72:     ON_WM_PAINT()
73: END_MESSAGE_MAP()
74:
```

```
75: // Construct the CWindowApp's m_pMainWnd data member
76: BOOL CWindowApp::InitInstance()
77: {
78:    m_pMainWnd = new CAppWindow();
79:    m_pMainWnd->ShowWindow(m_nCmdShow);
80:    m_pMainWnd->UpdateWindow();
81:    return TRUE;
82: }
83:
84: // application's constructor initializes and runs the app
85: CWindowApp WindowApp;
```

图 15.1 中显示了程序 WINDOW1.EXE 的一个运行范例。

图 15.1　程序 WINDOW1.EXE 的一个运行范例

编译和运行该程序，我们可以看到，当窗口建立时，将出现文本行。只要左上角没被其他窗口遮挡，那么当改变窗口大小，使之最大化、最小化、复原为正常大小时，文本行总是可见的(至少是部分可见的)。单击鼠标左按钮以显示一个消息框，将此消息框拖到文本行上并松开左按钮，然后又将消息框拖离文本位置，我们可以欣然地看到，文本行又出现了——由此说明 OnPaint 是一直工作的。

分　析

清单 15.3 中列出了程序 WIDOW1.CPP 的源代码。其中对本应用程序来说最重要的部分是 CAppWindow 类和它的成员函数，该窗口类定义了一个构造函数和三个成员函数，名为 OnLButtonDown，OnClose 和 OnPaint。构造函数利用 EXITMENU 菜单资源建立一个具有缺省大小和位置的、标题为"A Simple Read-Only Text Window"的窗口。

我们最关注的是第 53 到 65 行定义的 OnPaint 函数。本函数定义了两个常量 MAX_LINES 和 LINE_INCR、串变量 s、设备描述表实例 dc 和布尔型变量 ok。在第 61 到 64 行，OnPaint 函数用 for 循环显示文本行。每个循环迭代执行两个语句，第一个语句位于第 62 行，它调用函数 sprintf(其原型在头文件 STDIO.H 中)以建立格式化输出的映像，并保存在变量 s 中。第 63 行的语句中调用了 TextOut 函数。注意函数 TextOut 的参数取值，前两个为 0(对应于参数 x)和 i* LINE_INCR(对应于参数 Y)，这些值使得文本从窗口的左边缘开始显示。常量 LINE_INCR 代表一行的总高度，其中包括显示字符的高度和空白线。第 3 和第 4 个参数的取值分别为字串变量 s 和它的长度 strlen(s)值。

15.2 文本的滚动

窗口的特性之一是：如果当前屏幕的可视部分不能容纳所要显示的信息时，可通过滚动来显示未见部分。滚动条是窗口的可视组件，用它可以滚动显示窗口的内容。一个窗口可以有一个垂直滚动条、一个水平滚动条，或两者都有。一个滚动条在每端都有一个箭头框，拥有一个滚动游标(scroll thumb)，箭头框允许你向两边或两头滚动窗口的内容。在今天的课程中，将学习如何滚动一个含有用设备描述表对象绘出了文本的窗口。

新 词

滚动游标(scroll thumb)

滚动游标有两个作用：第一，它表示当前位置在所察看的信息的全部宽度或长度中的相对位置；第二，当用鼠标拖动滚动游标时，它将在所察看的信息中快速地推进。

今天这一课开始时提到的滚动效果，可以由可视界面和内部“引擎”支持。只要在成员函数 Create 的 dwStyle 参数中加入 WS_VSCROLL 和 WS_HSCROLL 风格，就可以很方便地在窗口中加入可视滚动条。例如，要建立一个同时拥有垂直和水平滚动条的窗口，可用如下的语句：

```
// both vertical and horizontal scroll bars
AStyle = WS_OVERLAPPEDWINDOW ¦WS_VSCROLL ¦WS_HSCROLL
```

常量 WS_VSCROLL 和 WS_HSCROLL 增加滚动条的可见范围。这些功能由函数 OnVscroll，OnHscroll，OnPaint，OnSize，SetScrollRange，SetScrollPos 和 ScrollWindow 支持，所有这些函数都从类 CWnd 继承而来。自然，需要在应用程序中重写前 4 个成员函数，以实现定制的滚动特性。

成员函数 SetScroll 设定垂直和水平滚动条滚动的最大和最小范围(以设备单位计)，成员函数 SetScrollPos 设定滚动条的位置，成员函数 ScrollWindow 水平或垂直滚动窗口的内容。成员函数 OnVscroll 响应 WM_VSCROLL 消息，垂直滚动一滚动条；成员函数 OnHScroll 则响应 WM_HSCROLL 消息，水平滚动一滚动条。

语 法

函数 SetScrollRange

函数 SetScrollRange 声明如下：

```
void SetScrollRange(int nBar, int nMinPos, int nMaxPos,
                    BOOL bRedraw = TRUE);
```

参数 nBar 是一数值开关，指明所要设定的滚动条，它可以是以下的预定义值之一：

·SB_VERT　设定垂直滚动条的范围
·SB_HORZ　设定水平滚动条的范围
·SB_CTL　设定滚动条控制的范围

参数 nMinPos 和 nMaxPos 定义了滚动条范围。如果给各参数都赋值 0，则滚动条被隐藏。滚动条的范围值不能超过 int 型的最大值，即 32767。布尔型参数 bRedraw 则指示当设定新的滚动条范围时是否重绘滚动条，缺省时滚动条将重绘，如取值为 FALSE，滚动条将不重绘。

示　例：

```
// set range for vertical scroll bar
SetScrollRange(SB_VERT, 0, 250);

// set range for horizontal scroll bar
SetScrollRange(SB_HORZ, 0, 500);
```

语　法

函数 SetScrollPos

函数 SetScrollPos 声明如下：

```
int SetScrollPos(int nBar, int nPos, BOOL bRedraw = TRUE);
```

参数 nBar 指定要设置的滚动条，可以为如下的预定义值之一：

SB_VERT　设置垂直滚动条位置
SB_HORZ　设置水平滚动条位置
SB_CTL　设置滚动条控制位置

参数 nPos 指定滚动条的新位置，它的值必须在函数 SetScrollRange 所设定范围之内。布尔型参数 bRedraw 指示当设定新的滚动范围时是否重绘滚动条，缺省时将重绘滚动条，如取值为 FALSE，滚动条将不重绘。函数 SetScrollPos 返回滚动条先前的位置。

示　例：

```
PrevPos = SetScrollPos(SB_VERT,75);
```

语　法

函数 ScrollWindow

函数 ScrollWindow 声明如下：

```
void ScrollWindow(int xAmount, int yAmount,
                  LPRECT lpRect = NULL,
                  LPRECT lpClipRect = NULL);
```

参数 xAmount 指定水平滚动数值，取值为正时窗口向右滚动，取负值时则向左滚动。yAmount 指定垂直滚动数值，取值为正时窗口向下滚动，取负值时则向上滚动。参数 lpRect 是指向定义了要滚动的客户区的位置的 CRect 或 RECT 变量的指针，它取缺省值 NULL 时整个客户区都滚动。参数 lpClipRect 是指向定义了要滚动的剪裁矩形域的 CRect 或 RECT 变量的指针，取缺省值 NULL 时滚动矩形内没有剪裁区。调用函数 ScrollWindow 将产生 WM_PAINT 消息，由成员函数 OnPaint 进行处理，因此必须在

MFC 应用程序中定义成员函数 OnPaint。

示　例:

```
ScrollWindow(nDeltaX, nDeltaY);
```

语　法

函数 OnVScroll

函数 OnVScroll 声明如下:

```
void OnVScroll(UINT nSBCode, UINT nPos, CWnd pScrollBar);
```

nSBCode 为指定垂直滚动请求的一个整型码。表 15.1 中列出了各种滚动请求的预定义值。

参数 nPos 指定滚动游标的位置。该值仅在参数 nSBCode 为 SB_THUMBPOSITION 或 SB_THUMBTRACK 时有效。参数 pScrollBar 为指向发送 WM_VSCROLL 消息的控制的指针。

示　例:

```
OnVScroll(SB_LINEDOWN, 23, NULL);
```

表 15.1　垂直滚动请求的预定义常量

值	含　义
SB_BOTTOM	滚到底部。
SB_ENDSCROLL	停止滚动。
SB_LINEDOWN	向下滚动一行。
SB_LINEUP	向上滚动一行。
SB_PAGEDOWN	向下滚动一页。
SB_PAGEUP	向上滚动一页。
SB_THUMBPOSITION	滚动到 nPos 位置。
SB_THUMBTRACK	拖着滚动条到 nPos 位置。
SB_TOP	滚动到最顶上。

语　法

函数 OnHScroll

函数 OnHScroll 声明如下:

```
void OnHScroll(UINT nSBCode, UINT nPos, CWnd* pScrollBar);
```

nSBCode 为指定水平滚动请求的整型码。表 15.2 中列出了各种滚动请求的预定义常量。参数 nPos 指定滚动游标的位置,它仅在参数 nSBCode 为 SB_THUMBPOSITION 时有效。参数 pScrollBar 为指向发送 WM_HSCROLL 消息的控制的指针。

示　例:

```
OnHScroll(SB_LINEDOWN, 14, NULL);
```

表 15.2　水平滚动请求的预定义常量

值	含　义
SB_BOTTOM	滚动到最右边。
SB_ENDSCROLL	停止滚动。
SB_LINEDOWN	向下滚动一行。
SB_LINEUP	向上滚动一行。
SB_PAGEDOWN	向下滚动一页。
SB_PAGEUP	向上滚动一页。
SB_THUMBPOSITION	滚动到 nPos 位置。
SB_TOP	滚动到最左边。

成员函数 OnSize 响应窗口大小发生改变时所发送的 WM_SIZE 消息。函数 OnSize 负责更新滚动参数(或大小值)。

语　法

函数 OnSize

函数 OnSize 声明如下:

```
void OnSize(UINT nType, int nCx, int nCy);
```

参数 nType 指示改变大小请求的类型。表 15.3 中列出了各种改变大小请求的预定义值。参数 nCx 和 nCy 指定客户区的新宽度和新高度。

示　例:

```
OnSize(SIZEFULLSCREEN, 13, 55)
```

表 15.3　改变大小的请求的预定义值

值	含　义
SIZEFULLSCREEN	窗口最大化。
SIZEICONIC	窗口最小化。
SIZENORMAL	窗口恢复正常大小。
SIZEZOOMHIDE	当另一窗口最大化时,发送消息给弹出窗口。
SIZEZOOMSHOW	当另一窗口恢复原先大小时,发送消息给弹出窗口。

当请求重绘窗口客户区时,不带参数的成员函数 OnPaint 响应 WM_PAINT 消息。函数 OnPaint 负责显示窗口更新过的内容。

15.3　滚 动 窗 口

本节提供了一个定义一般可滚动窗口的程序。WINDOW2.EXE 有两个主菜单项项 Exit 和 Char Sets。第二个菜单项是拥有三个选项的弹出菜单:Set1, Set2 和 Set3。这些选项使得文本行具有不同的行间距和最大行数。在所有的三种设置中,文本字体都一样。开始时,应用程序窗口的客户区清除,这样就必须选择三个字符设置中的一个。应用程序也支持用光标移动键滚动文本,Home 键和 End 键分别垂直滚动到头部和尾部。PageUp 键

和 PageDown 键分别向上或向下滚动一行。左和右箭头键分别向左和向右水平移动一页。

下面的程序示意了有关窗口滚动的两个要点:

1. 定义一个具有附加成员函数的滚动窗口类,这些附加成员函数用来管理赋值、存取和使用有关滚动的数据。
2. 用给定的值控制垂直滚动,并根据当前的窗口尺寸控制水平滚动。

我们来看一个一般的文本滚动窗口应用程序的代码。清单 15.4 和 15.5 中列出了头文件 WINDOW2.H 和资源文件 WINDOW2.RC。清单 15.6 中则包含了 WINDOW2.CPP 程序的源代码。

在目录\MSVC\VC21DAY\下建立子目录 WINDOW2,并把所有的项目文件放入新的目录。.MAK 文件将包含文件 WINDOW2.CPP, WINDOW2.RC 和 WINDOW2.DEF。最后一个文件与第 14 天的清单 14.2 中的 MINWINAP.DEF 文件相似。

键　入　清单 15.4　头文件 WINDOW2.H 的源代码

```
1: # define CM_EXIT (WM_USER + 100)
2: # define CM_SET1 (WM_USER + 101)
3: # define CM_SET2 (WM_USER + 102)
4: # define CM_SET3 (WM_USER + 103)
```

键　入　清单 15.5　资源描述文件 WINDOW2.RC

```
1: # include < windows.h>
2: # include < afxres.h>
3: # include "window2.h"
4:
5: MAINMENU MENU LOADONCALL MOVEABLE PURE DISCARDABLE
6: BEGIN
7:    MENUITEM "E&xit", CM_EXIT
8:    POPUP "&Char Sets"
9:    BEGIN
10:       MENUITEM "Set &1", CM_SET1
11:       MENUITEM "Set &2", CM_SET2
12:       MENUITEM "Set &3", CM_SET3
13:    END
14: END
```

键　入　清单 15.6　程序文件 WINDOW2.CPP 的源代码

```
1: # include < afxwin.h>
2: # include < stdio.h>
3: # include < stdlib.h>
```

```
4: # include < string. h>
5: # include "window2. h"
6:
7: // delcare the initial arguments for the window class constructor
8: const DefYMax = 100;
9: const DefXLine = 1;
10: const DefYLine = 1;
11: const DefXPage = 5;
12:
13: // Define a window class derived from CFrameWnd
14: class CAppWindow: public CFrameWnd
15: {
16: public:
17:    CAppWindow();
18:
19: protected:
20:    // declare data members to store character and page metrics
21:    int nCharWidth;
22:    int nCharHeight;
23:    int nXLine;
24:    int nYLine;
25:    int nXPage;
26:    int nYPage;
27:    int nVScrollMax;
28:    int nHScrollMax;
29:    int nVScrollPos;
30:    int nHScrollPos;
31:    int nxClient;
32:    int nyClient;
33:    BOOL bNoPaint;
34:    BOOL bNoInitPaint;
35:
36:    // handle vertical scrolling
37:    afx- msg void OnVScroll(UINT nSBCode, UINT nPos,
38:                            CScrollBar* pScrollBar);
39:    // handle horizontal  scrolling
40:    afx- msg void OnHScroll(UINT nSBCode, UINT nPos,
41:                            CScrollBar* pScrollBar);
42:    // handle pressing keys
43:    afx- msg void OnKeyDown(UINT nChar, UINT nRepCnt, UINT Flags);
44:    // handle selecting character set # 1
45:    afx- msg void OnSet1();
46:    // handle selecting character set # 2
47:    afx- msg void OnSet2();
48:    // handle selecting character set # 3
49:    afx- msg void OnSet3();
50:    // handle the Exit menu option
51:    afx- msg void OnExit();
52:    // member functions to set the scroll parameters
53:    void SetCharHeight(int nCharHeightVal);
54:    void SetLineSize(int nXLineVal, int nYLineVal);
55:    void SetVScrollMax(int nVScrollMaxVal);
```

```
56:    void SetPageSize(int nXPageVal);
57:    // member functions to get the scroll parameters
58:    void GetCharHeight(int& nCharWidth, int& nCharHeightVal);
59:    void GetLineSize(int& nXLineVal, int& nYLineVal);
60:    void GetScrollMaxSize(int& nHScrollMaxVal,
61:                          int& nVScrollMaxVal);
62:    void GetRangeSize(int& nXPageVal, int& nYPageVal);
63:    // handle closing the window
64:    afx_msg void OnClose();
65:    // handle resizing the window
66:    afx_msg void OnSize(UINT nTypes, int nCx, int nCy);
67:    // handle painting the window
68:    afx_msg void OnPaint();
69:
70: private:
71:    void makeSet(int nX, int nY);
72:    int GetVScrollPos(int nVpos)
73:    { return int(long(nVpos) * long(nYClient) / nVScrollMax); }
74:
75:    // message map macro
76:    DECLARE_MESSAGE_MAP();
77: };
78:
79: // Define an application class derived from CWinApp
80: class CWindowApp: public CWinApp
81: {
82: public:
83:    virtual BOOL InitInstance();
84: };
85:
86: CAppWindow::CAppWindow()
87: {
88:    DWORD AttrStyle = WS_OVERLAPPEDWINDOW ©¦WS_VSCROLL
89:                      ©¦WS_HSCROLL;
90:    UINT ClassStyle = CS_VREDRAW ©¦CS_HREDRAW;
91:    const char FAR* lpszClassName;
92:    HBRUSH hColor = (HBRUSH)(COLOR_HIGHLIGHTTEXT+ 1);
93:
94:    // disable initial painting
95:    bNoInitPaint = TRUE;
96:    // get the registration class name
97:    lpszClassName = AfxRegisterWndClass(ClassStyle, NULL, hColor,
98:                                        NULL);
99:    // create the window
100:    Create(lpszClassName, "A General Text Scroller Window",
101:           AttrStyle, rectDefault, NULL, "MAINMENU");
102:    // set the vertical and horizontal line heights
103:    SetLineSize(DefXLine, DefYLine);
104: }
105:
106: void CAppWindow::OnVScroll(UINT nSBCode, UINT nPos,
107:                            CScrollBar* pScrollBar)
```

```
{
   short nScrollInc;
   int nNewPos;

   switch (nSBCode) {
      case SB_TOP: // move to the top
        nScrollInc = - nVScrollPos;
        break;
      case SB_BOTTOM: // move to the bottom
         nScrollInc = nVScrollMax - nVScrollPos;
        break;
      case SB_LINEUP: // move one line up
           nScrollInc = - nYLine;
        break;
      case SB_LINEDOWN: // move one line down
           nScrollInc = nYLine;
        break;
      case SB_PAGEUP: // move one page up
           nScrollInc = - nYPage;
        break;
      case SB_PAGEDOWN: // move one page down
           nScrollInc = nYPage;
        break;
      case SB_THUMBPOSITION: // track the thumb box
        nScrollInc = nPos - nVScrollPos;
        break;
      default:
        nScrollInc = 0;
   }
   // calculate new vertical thumb position so that:
   //      0 <= nNewPos <= nVScrollMax
   nNewPos = max(0, min(nVScrollPos + nScrollInc, nVScrollMax));
   // adjust scroll increment
   nScrollInc = nNewPos - nVScrollPos;
   // is nScrollInc not zero?
   if (nScrollInc) {
      nVScrollPos = nNewPos; // update vertical thumb position
      // scroll the window
      ScrollWindow(0, - nScrollInc * nCharHeight);
      // move the thumb box
      SetScrollPos(SB_VERT, GetVScrollPos(nVScrollPos);
      // repaint the window
      UpdateWindow();
   }
}

void CAppWindow::OnHScroll(UINT nSBCode, UINT nPos,
                           CScrollBar* pScrollBar)
{
   short nScrollInc;
   int nNewPos;

```

```
160:    switch (nSBCode) {
161:       case SB_LINEUP: // move to the left by one character
162:          nScrollInc = - nXLine;
163:         break;
164:       case SB_LINEDOWN: // move to the right by one character
165:          nScrollInc = nXLine;
166:         break;
167:       case SB_PAGEUP: // move to the left
168:          nScrollInc = - nXPage;
169:         break;
170:       case SB_PAGEDOWN: // move to the right
171:          nScrollInc = nXPage;
172:         break;
173:       case SB_THUMBPOSITION: // track the thumb box
174:          nScrollInc = nPos - nHScrollPos;
175:         break;
176:       default:
177:         nScrollInc = 0;
178:    }
179:    // calculate new horizontal thumb position so that:
180:    //       0 <= nNewPos <= nHScrollMax
181:    nNewPos = max(0, min(nHScrollPos + nScrollInc, nHScrollMax));
182:    // adjust scroll increment
183:    nScrollInc = nNewPos - nHScrollPos;
184:    // is nScrollInc not zero?
185:    if (nScrollInc) {
186:       nHScrollPos = nNewPos; // update thumb position
187:       // scroll window
188:       ScrollWindow( - nScrollInc * nCharWidth, 0);
189:       // set the thumb box position
190:       SetScrollPos(SB_HORZ, nHScrollPos * nCharWidth);
191:       // repaint the window
192:       UpdateWindow();
193:    }
194: }
195:
196: void CAppWindow::OnKeyDown(UINT nChar, UINT nRepCnt, UINT Flags)
197: {
198:    switch(nChar) {
199:       case VK_HOME:
200:          OnVScroll(SB_TOP, 0, NULL);
201:          break;
202:       case VK_END:
203:          OnVScroll(SB_BOTTOM, 0, NULL);
204:          break;
205:       case VK_PRIOR:
206:          OnVScroll(SB_PAGEUP, 0, NULL);
207:          break;
208:       case VK_NEXT:
209:          OnVScroll(SB_PAGEDOWN, 0, NULL);
210:          break;
211:       case VK_UP:
```

```
212:          OnVScroll(SB_LINEUP, 0, NULL);
213:          break;
214:       case VK_DOWN:
215:          OnVScroll(SB_LINEDOWN, 0, NULL);
216:          break;
217:       case VK_LEFT:
218:          OnHScroll(SB_PAGEUP, 0, NULL);
219:          break;
220:       case VK_RIGHT:
221:          OnHScroll(SB_PAGEDOWN, 0, NULL);
222:          break;
223:    }
224: }
225:
226: void CAppWindow::OnSet1()
227: { makeSet(0, DefYMax); }
228:
229: void CAppWindow::OnSet2()
230: { makeSet(-5, 2*DefYMax); }
231:
232: void CAppWindow::OnSet3()
233: { makeSet(-10, DefYMax/2); }
234:
235: void CAppWindow::makeSet(int nX, int nY)
236: {
237:    // enable painting
238:    bNoInitPaint = FALSE;
239:    // set default character height
240:    SetCharHeight(nX);
241:    // set maximum vertical scrolling range
242:    SetVScrollMax(nY);
243:    // set the page size
244:    SetPageSize(DefXPage);
245:    // clear the window client area
246:    InvalidateRect(NULL, TRUE);
247: }
248:
249: void CAppWindow::OnExit()
250: { SendMessage(WM_CLOSE); }
251:
252: void CAppWindow::SetCharHeight(int nCharHeightVal)
253: {
254:    TEXTMETRIC tm;
255:    CRect r;
256:    CDC* dc = GetDC();
257:
258:    // get the text metrics
259:    dc->GetTextMetrics(&tm);
260:    ReleaseDC(dc);
261:    // get the client area coordinates
262:    GetWindowRect(&r);
263:    nxClient = r.right - r.left;
```

```
264:    nyClient = r.bottom - r.top;
265:    // obtain the character height
266:    if (nCharHeightVal == 0)
267:       nCharHeight = tm.tmHeight + tm.tmExternalLeading;
268:    else if (nCharHeightVal > 0)
269:       nCharHeight = nCharHeightVal;
270:    else
271:       nCharHeight = tm.tmHeight + tm.tmExternalLeading -
272:                     nCharHeightVal;
273:    // set the range of the vertical scroll bar
274:    SetScrollRange(SB_VERT, 0, nyClient, TRUE);
275:    // get the character width
276:    nCharWidth = tm.tmAveCharWidth;
277:    // set the range of the horizontal scroll bar
278:    SetScrollRange(SB_HORZ, 0, nxClient, TRUE);
279: }
280:
281: void CAppWindow::SetLineSize(int nXLineVal, int nYLineVal)
282: {
283:    // set the vertical and horizontal line heights
284:    nXLine = nXLineVal;
285:    nYLine = nYLineVal;
286: }
287:
288: void CAppWindow::SetVScrollMax(int nVScrollMaxVal)
289: {
290:    // set the vertical and horizontal limits
291:    nVScrollMax = nVScrollMaxVal;
292:    // set the initial line position
293:    nVScrollPos = 0;
294:    // set the position of the vertical scroll bar
295:    SetScrollPos(SB_VERT, 0, TRUE);
296:    // calculate the maximum number of characters
297:    nHScrollMax = nxClient / nCharWidth;
298:    // set the initial column position
299:    nHScrollPos = 0;
300:    // set the position of the horizontal scroll bar
301:    SetScrollPos(SB_HORZ, 0, TRUE);
302:    UpdateWindow();
303: }
304:
305: void CAppWindow::SetPageSize(int nXPageVal)
306: {
307:    // set the vertical and horizontal page up/down sizes
308:    nXPage = nXPageVal;
309:    nYPage = nyClient / nCharHeight;
310: }
311:
312: void CAppWindow::GetCharHeight(int& nCharHeightVal,
313:                                int& nCharWidthVal)
314: {
315:    nCharHeightVal = nCharHeight;
```

```
316:    nCharWidthVal =  nCharWidth;
317: }
318:
319: void CAppWindow::GetLineSize(int& nXLineVal, int& nYLineVal)
320: {
321:    nXLineVal =  nXLine;
322:    nYLineVal =  nYLine;
323: }
324:
325: void CAppWindow::GetScrollMaxSize(int& nHScrollMaxVal,
326:                                   int& nVScrollMaxVal)
327: {
328:    nHScrollMaxVal =  nHScrollMax;
329:    nVScrollMaxVal =  nVScrollMax;
330: }
331:
332: void CAppWindow::GetPageSize(int& nXPageVal, int& nYPageVal)
333: {
334:    nXPageVal =  nXPage;
335:    nYPageVal =  nYPage;
336: }
337:
338: void CAppWindow::OnClose()
339: {
340:    if (MessageBox("Want to close this application",
341:          "Query", MB_YESNO ©¦MB_ICONQUESTION) ==  IDYES)
342:       DestroyWindow();
343: }
344:
345: void CAppWindow::OnSize(UINT nTypes, int nCx, int nCy)
346: {
347:       // disable painting?
348:       bNoPaint =  nTypes ==  SIZEICONIC ©¶nTypes ==  SIZEZOOMHIDE;
349:       // exit if painting is disabled
350:       if (bNoPaint ©¶bNoInitPaint) return;
351:
352:       // update the client area
353:       nxClient =  nCx;
354:       nyClient =  nCy;
355:       // update the vertical page size
356:       nYPage =  nyClient / nCharHeight;
357:       // set the new vertical scroll bar range
358:       SetScrollRange(SB_VERT, 0, nyClient, FALSE);
359:       // set the new vertical thumb box position
360:       SetScrollPos(SB_VERT, GetVScrollPos(nVScrollPos), TRUE);
361:       // calculate the new maximum  width
362:       nHScrollMax =  max(1, nxClient / nCharWidth);
363:       // set the new horizontal scroll bar range
364:       nHScrollPos =  min(nHScrollPos, nHScrollMax)
365:       // set the new horizontal scroll bar range
366:       SetScrollRange(SB_HORZ, 0, nxClient, FALSE);
367:       // set the new horizontal thumb box position
```

```
368:     SetScrollPos(SB_HORZ, nHScrollPos * nCharWidth, TRUE);
369:     // clear and repaint the window
370:     Invalidate(TRUE);
371: }
372:
373: void CAppWindow::OnPaint()
374: {
375:   CPaintDC dc(this);
376:   char aStr[80];
377:   char* pStr;
378:
379:   // exit if OnPaint is disabled
380:   if (bNoPaint ©©þbNoInitPaint) return;
381:
382:   // draw lines of text
383:   for (int i = 0, j = i + nVScrollPos;
384:         i < nYPage && j < nVScrollMax;
385:         i++, j++) {
386:     sprintf(aStr, "This is line number %d   ", j+1);
387:     // is the line visible?
388:     if (nHScrollPos < (int)(strlen(aStr))) {
389:       pStr = aStr + nHScrollPos;
390:       dc.TextOut(0, i * nCharHeight, pStr, strlen(pStr));
391:     }
392:   }
393: }
394:
395: BEGIN_MESSAGE_MAP(CAppWindow, CFrameWnd)
396:     ON_WM_VSCROLL()
397:     ON_WM_HSCROLL()
398:     ON_WM_KEYDOWN()
399:     ON_COMMAND(CM_SET1,OnSet1)
400:     ON_COMMAND(CM_SET2,OnSet2)
401:     ON_COMMAND(CM_SET3,OnSet3)
402:     ON_COMMAND(CM_EXIT,OnExit)
403:     ON_WM_CLOSE()
404:     ON_WM_SIZE()
405:     ON_WM_PAINT()
406: END_MESSAGE_MAP()
407:
408: // Construct the CWindowApp's m_pMainWnd data member
409: BOOL CWindowApp::InitInstance()
410: {
411:   m_pMainWnd = new CAppWindow();
412:   m_pMainWnd->ShowWindow(m_nCmdShow);
413:   m_pMainWnd->uPdateWindow();
414:   return TRUE;
415: }
416:
417: // application's constructor initializes and runs the app
418: CWindowApp WindowApp;
```

图 15. 2 中显示了程序 WINDOW2. EXE 的一个运行范例。

图 15. 2　程序 WINDOW2. EXE 的一个运行范例

分　析

清单 15. 6 中的程序定义了一组全局常量, 作为一些字符和页尺寸的缺省(或初始)值。本程序定义了两个类: 一个应用程序类和一个窗口类, 后者是最有用的一个。

窗口类 CAppWindow 定义了若干数据成员和成员函数(公有、保护、以及私有的)。数据成员位于第 20 到 34 行, 用于管理文本的显示, 它们又可以分为以下几个小组:

- 成员 nCharHeight 和 nCharWidth 分别存放字符高度和宽度, 这些成员一般从 Windows 文本尺寸中获取数据, 但是本程序代码在数据成员 nCharHeight 中包括了行间距, 因而窗口类允许你影响字符的高度。
- 成员 nXLine 和 nYLine 分别存放在水平和垂直行滚动中用的字符数和行数。这两数据成员一般都取值为 1。
- 成员 nXPage 和 nYPage 分别为水平和垂直页滚动中使用的字符数和行数。成员 nXPage 只需赋一个值, 而成员 nYPage 的值则是由客户窗口的高度除以字符高度而得到的。因此, 垂直页的大小视窗口的大小、字符高度和字符空距而定。
- 成员 nVScrollMax 和 nHScrollMax 分别存放垂直和水平滚动条范围的最大值。成员 nVScrollMax 只需赋一值, 而成员 nHScrollMax 则是通过计算客户窗口的宽度和字符宽度的比值而得到的。
- 成员 nVScrollPos 和 nHScrollPos 分别存放垂直和水平滚动条的位置。这些成员的值随着相应滚动条的滚动而改变。
- 成员 nCxClient 和 nCyClient 分别存放客户区的宽度和高度。这些成员的值由成

员函数 OnSize 进行更新。

- 成员 bNoPaint 是一个标志。在窗口被最小化或隐藏在一个最大化窗口的下面时，它可以阻止窗口重画。
- 布尔型成员 bNoInitPaint 禁止在从 Char Sets 菜单选择了字符之前进行绘画。

类的构造函数位于第 86 到 104 行，主要负责进行初始化和建立应用程序窗口，具体如下：

- 定义若干已初始化的变量，用于窗口类(类别)注册处理。
- 将布尔型数据成员 bNoInitPaint 赋值为 TRUE，初始化为禁止绘画。
- 调用全局函数 AfxRegisterWndClass，获得注册类(类别)名，这样可以对窗口的建立进行细微的调节。
- 建立一窗口实例。
- 调用成员函数 SetLineSize 设置行大小。

> **注　意**　类 CAppWindow 中声明了私有成员函数 GetVScrollPos。此函数位于第 72 到 73 行，它把所显示的第一行中的数值转换成设备单位。此函数对于正确定位滚动条的滚动游标很重要。要记住窗口中所画的文本和垂直滚动条不是以同样数目的设备单位滚动的。

第 106 到 152 行定义了成员函数 OnVScroll，它响应 WM_VSCROLL 消息，移动垂直滚动条。该函数定义了两个局部变量：nScrollInc 和 nNewPos。变量 nScrollInc 保存滚动的行数，变量 nNewPos 存放所期望的滚动条的新位置。函数 OnVScroll 进行如下操作：

- 利用 switch 语句判断垂直滚动请求的类型(注意，行和页滚动请求现在使用的是数据成员 nYLine 和 nYPage，而不是文字常量。)
- 计算所期望的滚动条的新位置，使用函数 min 和 max(在 stdlib.h 中定义)确保新位置在范围 0 到 nVScrollMax 之间。
- 计算调整过的滚动行数，并保存在 nScrollInc 中。
- 当 nScrollInc 的调整值非零时，垂直滚动窗口(if 语句赋一个新值给 nVScrollPos，调用函数 ScrollWindow 垂直滚动窗口；调用函数 SetScrollPos 和 GetVScrollPos 更新垂直滚动条位置；调用函数 UpdateWindow 重绘窗口)。

请注意，窗口垂直滚动了(- nScrollInc * nCharHeight)设备单位，垂直滚动条移到 GetVScrollPos(nVScrollPos)位置。

第 154 到 194 行定义了成员函数 OnHScroll，它响应 WM_HSCROLL 消息，水平滚动窗口。此函数代码与函数 OnVScroll 相似，不同之处在于水平滚动条移到 nHScrollPos * nCharWidth 位置，这个值是利用与函数 OnVScroll 中的截然不同的表达式算出的。

第 196 到 224 行定义了成员函数 OnKeyDown，它处理 WM_KEYDOWN 消息，判断是否和怎样响应键盘输入。本函数用一个 switch 语句去处理 Home，End，PageUp，PageDown 以及上、下、左、右箭头键。各个 case 标号中的语句或者调用函数 OnVScoll 或 OnHScroll，这些调用使用预定义的 SB_xxxx 常量来确定滚动请求的类型。所有这些调用

中，函数 OnVScroll 和 OnHScroll 的参数 nPos 和 pScrollBar 都分别取值为 0 和 NULL。

窗口类定义了三个相似的成员函数，即 OnSet1，OnSet2 和 OnSet3，用于装入不同的行设置。这些函数在第 226 到 233 行定义，它们依次以不同的参数取值调用成员函数 makeset 完成必要的任务。函数 makeset 定义在第 235 至 247 行，它执行如下的操作：

- 将 bNoInitPaint 赋值为 FALSE，允许绘画；
- 设置字符高度；
- 设置垂直滚动条的最大限制；
- 设置水平页大小，三个 OnSetx 函数都取相同的参数值；
- 清除窗口和请求重画客户窗口区。

成员函数 SetCharHeight 在第 252 至 279 行定义，它设定字符高度并更新字符宽度。此函数只有一个参数：nCharHeightVal，这是因为只允许设置或影响字符高度。字符宽度将利用当前的文本宽度算出。该函数执行如下操作：

- 调用 GetTextMetrics 函数获得文本尺寸，调用时传递引用变元 tm，以获得当前的文本尺寸；
- 获得窗口的宽度和高度，并存储到成员 nCxClient 和 nCyClient 中；
- 获得字符高度；
- 设置垂直滚动条范围为 0 到 nCyClient；
- 利用 tm 结构体出的有关数据，计算字符高度；
- 设置水平滚动条范围从 0 到 nCxClient。

用 SetCharHeight 获得字符高度时有三种可能的情况：如 nCharHeightVal 取值为 0，函数只使用从局部结构 tm 获得的文本尺寸数据计算 nCharHeight 的值；如果 nCharHeightVal 为正值，函数只需将 nCharHeightVal 值赋给成员 nCharHeight；如果 nCharHeightVal 为负值，函数将把它与从 tm 结构获得的字符高度相加，换句话说，取值为负的 nCharHeightVal 将被解释为附加的行间距(用设备单位)。

> **警　告**　用本程序做试验时应注意：如果 nCharHeightVal 取值为正，函数将把 nCharHeightVal 赋值给成员 nCharHeight。

第 281 到 286 行定义了成员函数 SetLineSize，它设置垂直和水平行的大小。传给此函数的参数的值将被赋给相应的数据成员。

第 288 到 303 行定义了成员函数 SetVScrollMax，它设置垂直和水平滚动条的最大范围并将滚动条重置到 0 位置。该函数只有一个参数 nVScrollMaxVal，这意味着调用者只能将新值赋给数据成员 nVScrollMax。该函数执行如下操作：

- 将 nVScrollMaxVal 的值赋给数据成员 nVScrollMax；
- 将数据成员 nVScrollPos 赋值为 0；
- 滚动垂直滚动条到位置 0；
- 计算水平滚动条的最大范围，并将结果存在成员 nHScrollMax 中；

· 将数据成员 nHScrollPos 赋值为 0;
· 滚动水平滚动条到位置 0。

第 305 到 310 行定义了成员函数 SetPageSize, 它设置垂直和水平页大小。此函数只有一个参数 nXPageVal, 这意味着调用者只能将值赋给数据成员 nXPage。成员 nYPage 的新值是通过求客户窗口的高度与字符高度(包括附加间距)的比值而得到的。

在第 312 到 316 行, 窗口类声明了成员函数 GetCharHeight, GetLineSize, GetScrollMax 和 GetPageSize。这些函数利用引用参数返回所查询的各数据成员。虽然本程序未使用这些函数, 但考虑到用户自己设计的滚动窗口可能非常需要这类函数, 所以还是将它们包括进来了。

第 345 到 371 行定义了成员函数 OnSize, 它处理 WM- SIZE 请求并更新滚动条的范围和位置。函数 OnSize 执行如下的操作:

· 检验窗口能否被画出, 成员 bNoPaint 用来存储非画出状态(如果 bNoPaint 或 bNoInitPaint 数据成员为 TRUE 则该功能存在)。
· 把重置过大小的客户区的宽度和高度(由参数 ncX 和 ncY 传递其值)分别赋给数据成员 nCxClient 和 nCyClient。
· 更新 nYPage 中所存放的垂直页的大小(新值为客户窗口高度与字符高度的比值)。
· 调用 SetScrollPos 函数设定垂直滚动条的新位置, 利用 GetVScrollPos 成员函数将垂直行转换为设备单位。
· 调用函数 SetScrollPos 设定垂直滚动条的新位置。
· 计算新的最大宽度, 取 1 和客户区宽度与字符宽度的比值二者中的较大值。
· 设置水平滚动条位置成员 nHScrollPos 的新值, 取旧位置和最大字符数目中的较小者。
· 调用函数 SetScrollPos, 设定新的滚动条范围为 0 到 nCxClient。
· 调用函数 SetScrollPos, 设定新的水平滚动条位置。
· 调用函数 Invalidate 清除和重画用户区。

第 373 到 393 行定义了成员函数 OnPaint, 它响应 WM- PAINT 消息请求, 以便重画窗口。函数 OnPaint 的代码很简单、短小, 它不需要求当前可见行数目的语句, 这些信息可从数据成员 nYPage 中获得。注意循环终止条件中使用到了数据成员 nVScrollMax。

小　　结

今天这一课讨论了建立可显示固定的以及可滚动文本的窗口的机理, 主要包括以下内容:

· 显示信息的只读文本窗口是产生帮助画面的基础。
· 利用多用途成员函数 CDC::TextOut 在窗口中写文本。
· 用垂直和水平滚动条可建立可滚动窗口。窗口滚动既可用鼠标也可以用光标控制键。可滚动窗口类使用成员函数 OnVScroll, OnHScroll, OnPaint, OnSize 以及其

他一些函数来管理文本尺寸。

问　与　答

问: MFC 库中有可滚动窗口类吗?

答: 有。MFC 库中有 CScrollView 类, 它实现一个可滚动视区——一个其内容可滚动的子窗口。

问: 成员函数 OnPaint 需要用类似于命令按钮的可视控制去维护窗口吗?

答: 不用。只需写出函数 OnPaint 去维护画出文本和图形的窗口, 在带控制的窗口中, 不必声明自己的 OnPaint 函数版本。

作　　业

作业部分所提供的测验题有助于巩固理解所学内容, 通过练习可以积累运用所学知识的经验。要尽量弄懂测验和练习的答案后再继续下一天的课程。附录 B 的‘答案’中提供了解答。

测　验

1. 对还是错?
 成员函数 OnPaint 只在需要时重画窗口。
2. 对还是错?
 继承的 CWnd::OnVScroll 和 CWnd::OnHScroll 成员函数提供缺省的滚动特性。
3. 对还是错?
 如果在己的窗口类中声明成员函数 OnPaint, 则还必须声明成员函数 OnSize。
4. 对还是错?
 如果在程序 WINDOW2.CPP 中略去成员函数 OnKeyDown, 则垂直滚动特性将会丧失。

练　习

修改 WINDOW2.CPP, 更换成员函数 OnSet1, OnSet2 和 OnSet3 的参数取值。另外, 还可试着再增加一些 OnSetx 菜单选项以及相应的成员函数。

第16天

MFC 控制

与 Windows 应用程序进行交互对话经常要用到对话框,对话框中包含各种各样的控制,如列表框、编辑控制(也叫做编辑框)和下压按钮等。这些控制可包含在窗口中,但最常见的是在对话框中。今天以及随后 3 天的课程中,我们将学习窗口中的控制,并集中讨论这些控制的基本特性。第 20 天将介绍对话框,并看看控制是如何与这些框一起工作的,今天将学习以下几个专题:

- 静态文本控制
- 编辑控制
- 下压按钮控制

弄懂各种控制并掌握它们的行为和交互特性,可以帮助你实现更高级的交互式 Windows 应用程序。今天及随后的两次课程将讨论控制类的构造函数和有关的成员函数。

16.1 静态文本控制

静态文本控制提供一个带有静态文本的窗口或对话框。类 CStatic 可实现静态文本控制。下面我们来看构造函数及其成员。

新 词

静态文本(static text)

静态文本是应用程序用户不轻易以及不准备修改的文本,但静态文本并不意味着文本像刻在石头上一样不可更改。事实上,静态文本控制允许 MFC 应用程序根据需要更改文本。也可以指定文本为永久的、不可改变的。这些选择完全取决于你。

类 CStatic 是 CWnd 的派生类,它提供由显示区、显示文本和文本属性所定义的静态文本。在这三个部分中,只有显示文本可以在运行时改变。

语 法

函数 Create

成员函数 Create 声明如下:

```
BOOL Creat(const char FAR* lpText   // control text
           DWORD dwStyle            // control style
           const RECT& rect         // control area
           CWnd* pParentWnd         // parent window
           UINT nID = 0xffff);      // control ID
```

参数 lpText 指定静态文本控制的文本。参数 dwStyle 确定控制的风格,此风格的取

值一般包括 WS- CHILD 和 WS- VISIBLE 风格。表 16.1 中列出了静态文本的风格 SS-xxxx 的各种取值。建立静态文本控制时可包含一个或多个这样的风格，参数 rect 指定控制所占的矩形区。参数 pParentWnd 为指向父窗口的指针，参数 nID 指定控制的 ID。函数 Create 的缺省参数值为 0xffff，这是因为静态文本控制一般不发送通知消息给它的父窗口。函数调用成功返回 TRUE，否则返回 FALSE。

示　例:

```
CStatic* pText;
// create control
pText- > Create("Operand 1", SS- SIMPLE, r, NULL);
```

表 16.1　静态文本的风格值

值	含　义
SS- BLACKFRAME	指定一个其框架颜色与主窗口的颜色一致的框(在默认的 Windows 配色表中为黑色)。
SS- BLACKRECT	指定一个用与主窗口相同的颜色填充的矩形(在默认的 Windows 配色表中为黑色)。
SS- CENTER	使静态文本字符居中，文本是可擦除的。
SS- GRAYFRAME	指定其框架颜色与屏幕背景相同的框(在默认的 Windows 配色表中为灰色)。
SS- GRAYRECT	选择一个用与屏幕背景相同的颜色填充的矩形(在默认的 Windows 配色表中为灰色)。
SS- LEFT	表示文本左对齐，文本是可擦除的。
SS- LEFTNOWORDWRAP	表示左对齐文本，文本不可擦除。
SS- NOPREFIX	指出在静态文本字串中前面加“&”的字符不是热键指示字符，而是静态文本字符的一部分。
SS- RIGHT	选择右对齐文本，文本可擦除。
SS- SIMPLE	表示各静态文本字符在运行期间不能改变，静态文本显示在单行上，并忽略分行符。
SS- GRAYFRAME	指定其框架与窗口背景色具有相同颜色的框(在默认的 Windows 配色表中为白色)。
SS- GRAYRECT	选择用与窗口背景相同的颜色填充的矩形(在默认的 Windows 配色表中为白色)。

函数 Create 中的 lpText 指针所指向的字符串中可以包含字符“&”以便从直观上指定一个热键——要想实际支持热键，应用程序中还需要加载加速键表。热键字符显示时带有下画线。热键字符之前必须有一“&”字符。如果串中包含多个“&”字符，则只有最后一个起作用，其他的“&”字符将不被显示而忽略。如果确实要显示“&”字符，则需指定风格 SS-NOPREFIX，使用这种风格的代价是不能显示热键字符。

现在让我们来看看静态文本控制的在运行时可以改变的成分，即文本。如果在建立一个 CStatic 实例时指定 SS- SIMPLE 风格，则不能改变其文本。从这种意义上说，CStatic 的实例的确是刻在了石头上。类 CStatic 允许使用从类 CWnd 继承的函数

GetWindowTextLength，GetWindowText 和 SetWindowText 以设置、查询和清除静态文本的字符。

语　法

函数 GetWindowTextLength

无参数成员函数 GetWindowTextLength 返回控制的文本长度。

```
int GetWindowTextLength();
```

示　例:

```
int nLen;
CStatic* pText;
// create control...
nLen = pText->GetWindowTextLength();
```

语　法

函数 GetWindowText

用成员函数 GetWindowText 可以访问静态文本字符，函数 GetWindowText 声明如下:

```
int GetWindowText(LPSTR lpString, int nMaxCount) const;
```

参数 lpString 为指向接收静态文本字符拷贝的字符串的指针。参数 nMaxCount 指定要拷贝的静态文本字符的最大数目。函数返回拷贝到指针 lpString 所指向的字符串的实际字符数。

示　例:

```
const int MAX_LEN = 80;
char s[MAX_LEN + 1];
int nLen;
CStatic* pText;
// create control...
nLen = pText->GetWindowText(s, MAX_LEN);
```

语　法

函数 SetWindowText。

成员函数 SetWindowText 用新的字符串重写当前静态文本字符。函数 SetWindowText 声明如下:

```
void SetWindowText(const char FAR* lpString);
```

参数 lpString 为指向控制的新文本的指针，如果新文本是个空串，函数 SetWindowText 只是清除静态文本控制实例中的文本。

示　例:

```
char s[81] = "New Window text";
CStatic* pText;
// create control...
pText->SetWindowText(s);
```

16.2　编 辑 控 制

Microsoft 基本类提供了实现编辑控制的 CEdit 类。在前面的一些使用对话框的程

序中我们已经遇见过此控制。用编辑控制可以在输入对话框中输入和编辑文本。这一节我们将更详细地讨论类 CEdit 的功能，因为若要在 MFC 应用程序中实现定制的文本编辑，则必须对成员函数 CEdit 非常熟悉。

CEdit 类

类 CEdit 是类 CWnd 的派生类，它实现了一个通用编辑控制，不但支持单行和多行文本，还具有剪切、粘贴、删除和清除文本的能力。编辑控制也可取消最近的文本改变并且可与剪贴板交换文本. 类 CEdit 有一个构造函数和一个用于建立各种类实例的成员函数 Create。

语　法

函数 Create

函数 Create 声明如下:

```
BOOL Create(DWORD dwStyle,      // control style
            const RECT& rect,   // control area
            CWnd* pParentWnd,   // parent window
            UINT nID);          // control ID
```

参数 dwStyle 指定控制风格，它的取值一般包括 WS_CHILD 和 WS_VISIBLE 风格。表 16.2 中列出了一组 ES_xxxx 编辑控制风格，建立的编辑控制可包含一个或多个这样的风格。参数 rect 为控制占用的矩形区域。参数 pParentWnd 为指向父窗口的指针，参数 nID 指定控制的 ID，如果函数 Create 成功调用则返回 TRUE，否则返回 FALSE。

示　例:

```
const ID_EDIT_TXT;
dwEditStyle = WS_CHILD |WS_VISIBLE |WS_TABSTOP |WS_BORDER
              |WS_LEFT |ES_AUTOHSCROLL;
CRect r = CRect(100, 30);
CEdit* pEdit;
pEdit->Create(dwEditStyle, r, this, IS_EDIT_TXT);
```

CEdit 实例的典型风格有 WS_CHILD, WS_VISIBLE, WS_TABSTOP, ES_LEFT, ES_AUTOHSCROLL 和 WS_BORDER，这些风格产生带边框的编辑控制且显示的文本左对齐，如果边框内容纳不下全部文本，则可以通过水平滚动来显示。当建立一个多行编辑控制时，需要加入以下的风格: ES_MULTILINT, ES_AUTOVSCROLL, WS_HSCROLL 和 WS_VSCROLL，这些附加的缺省风格支持多行文本、使文本可竖直滚动，并包含竖直和水平滚动条。

表 16.2　编辑控制的风格值

值	含　义
ES_AUTOHSCROLL	当用户在行尾键入一个字符时，使文本能自动向右滚动 10 个字符；当用户按 Enter 键时，文本向左滚回。
ES_AUTOVSCROLL	当用户在可见文本的最末一行上按 Enter 键时，允许文本上翻一页。
ES_CENTER	使多行编辑控制中的文本居中。
ES_LEFT	文本左对齐。
ES_LOWERCASE	将用户键入的所有字母转换为小写。

值	含　义
ES_MULTILINE	指定能识别分行符(由回车和换行字符组成)的多行编辑控制。

续表

值	含　义
ES. NOHIDESEL	缺省情况下,当失去焦点时隐藏选中的文本,重获焦点时则显示选择项;禁止编辑控制恢复选中的文本。
ES. RIGHT	文本右对齐。
ES. UPPERCASE	将用户键入的所有字母转换为大写。

与剪贴板有关的编辑函数

类 CEdit 包含处理与剪贴板相关的编辑命令的一系列成员函数,这些编辑命令可通过典型的菜单选项来使用: Cut, Copy, Paste, Clear, Undo 和 Delete。表 16.3 中列出了 CEdit 成员函数及其用途,这些函数工作时使用具有 CF. TEXT 格式的剪贴板。

表 16.3　支持与剪贴板有关的编辑菜单命令的 CEdit 成员函数

成员函数	作　用
Cut	删除编辑控制的当前选中文本,并将其复制到剪贴板中。
Copy	将当前选中文本复制到剪贴板中。
Paste	将剪贴板中的文本插入到编辑控制中的当前光标位置。
Clear	删除当前选中的文本,不影响剪贴板。
Undo	取消编辑控制中最近一次对文本的改动。

编辑控制的查询

类 CEdit 有一系列文本查询成员函数,用这些函数可以保留全部或部分控制文本,还可以获得文本的有关统计信息(行数、行长等)。与查询有关的成员函数有:

- 无参数成员函数 GetWindowTextLength 返回编辑控制的文本长度,此成员函数从父类 CWnd 继承而来。
- 用成员函数 GetWindowText 可以访问编辑控制的字符,此函数也是从父类继承而来。
- 成员函数 GetLineCount 返回多行编辑控制的行数。
- 成员函数 LineFromChar 返回指定字符索引的行号。
- 成员函数 LineIndex 返回特定行的字符索引,该索引为编辑控制中到指定行号的文本长度。
- 成员函数 LineLength 返回特定的字符索引所对应的行的长度。
- 成员函数 GetSel 返回选中文本的起始和终止位置。起始的字符位置为选中文本的第一个字符的索引,终止位置为紧随选中文本之后的第一个字符的索引。
- 重载成员函数 GetLine 返回多行编辑控制中的一行。

语　法

函数 GetWindowText

函数 GetWindowText 声明如下:

```
int GetWindowText(LPSTR lpString, int nMaxCount) const;
```

参数 lpString 为指向接收静态文本字符的字符串的指针，参数 nMaxCount 指定要拷贝的静态文本字符的最大数目，函数返回拷贝到 lpString 所指向的字符串中的实际字符数。

示　例：

```
const MAX_SIZE = 80;
char s[MAX_SIZE + 1];
int nLen;
CEdit* pEdit;
// create control...
nLen = pEdit->GetWindowText(s, MAX_SIZE);
```

语　法

函数 GetLineCount

函数 GetLineCount 声明如下：

```
int GetLineCount() const;
```

示　例：

```
CEdit* pEdit;
// create control...
nLineCount = pEdit->GetLineCount();
```

注　意　在多行编辑控制中，要考虑软或硬分行符中的字符。硬分行符在每行结尾使用一对回车换行字符，软分行在分行处使用两个回车和一个换行字符。了解这些对于计算要处理的字符数目是很有用的。

语　法

函数 LineFromChar

函数 LineFromChar 声明如下：

```
int LineFromChar(int nIndex = -1) const;
```

本函数返回字符索引 nIndex 指定的行号，缺省的表达式为-1，要求返回下面这些值：

- 如有选择文本，函数返回第一选择字符所在行的行号。
- 如没选择文本，函数返回插入字符当前位置的行号。

示　例：

```
int nIdx = 3;
int nLineNum;
CEdit * pEdit;
// create control...
nLineNum = pEdit->LineFromChar(nIdex);
```

语　法

函数 LineIndex

函数 LineIndex 声明如下：

```
int LineIndex(int nLine = -1) const;
```

参数 nLine 指定行索引，表达式- 1 指定当前，本函数返回从第一页到指定项的字符总数，如果 nLine 表达式大于实际行数，函数返回- 1。

示　例：

```
CEdit* pEdit;
// create control...
nCharIndex = pEdit- > LineIndex();
```

语　法

函数 LineLength

函数 LineLength 声明如下：

```
int LineLength(int nLine = - 1) const;
```

参数 nLine 设定字符索引，当表达式为- 1 时，返回当前行的长度，在单行编辑控制中，函数 LineLength 返回编辑控制中的所有字符数。

示　例：

```
int nLineNum = 1;
int nLen;
CEdit* pEdit;
// create control...
nLen = pEdit- > LineLength(nLineNum);
```

语　法

函数 GetSel

函数 GetSel 声明如下：

```
DWORD GetSel() const;
```

本函数返回 32 位整数值，其中低字位为开始位置，高字位为结束位置，用预定义宏 HIWORD 和 LOWORD 取出高位和低位字。如果这两个字相等，就没有选择文本，因为两个字都指着当前插入位置。

示　例：

```
DWORD dwSelPos;
CEdit* pEdit
// create control...
dwSelPos = pEdit- > GetSel();
```

语　法

函数 GetLine

函数 GetLine 有两种形式，声明如下：

```
int GetLine(int nIndex, LPSRT lpBuffer) const;
int GetLine(int nIndex, LPSRT lpBuffer, int nMaxLength) const;
```

参数 nIndex 指明要检索的行的索引。lpBuffer 是用来存放检索行的缓存区，lpBuffer 的起始两个字节为行的长度。参数 nMaxLength 指定拷到缓存区 lpBuffer 的最大字节数。当使用 GetLine 的第二种形式时，函数拷贝 nMaxLength 的值到 lpBuffer 的头两个字节中。

示　例：

```
const MAX- STR = 80;
int nLineIndex;
```

```
int nNumChars;
char s[MAX_STR + 1];
CEdit* pEdit;
// create control...
nNumChars = pEdit->GetLine(nLineIndex, s, MAX_STR);
```

> **注　意**　因为 lpBuffer 不是 ASCIIZ 串，因此可以编写另一版本的 GetLine，以便返回 ASCIIZ 串。当然，新版本的 GetLIne 必须加入到 CEdit 的派生类中。

改变编辑控制

现在我们讨论 CEdit 的成员函数中有关改变编辑控制文本的部分，这些成员函数的操作包括将新文本写入到控制中、选中文本以及替换选中的文本等。

- 成员函数 SetWindowText，它是从父类继承而来的。该函数用新串中的字符重写当前编辑控制字符。
- 重载成员函数 SetSel 定义一字符块为新选中的文本。
- 成员函数 ReplaceSel 用新字符替换选中的文本。

语　法

函数 SetWindowText

函数 SetWindowText 声明如下：

```
Void SetWindowText (const char FAR* lpString);
```

参数 lpString 为指向控制的新文本的指针。如果新文本是空字符串，函数 SetWindowText 只是清除编辑控制实例中的文本。

示　例：

```
CEdit * pEdit;
// create control...
pEdit->SetWindowText("Calculate");
```

语　法

函数 SetSel

函数 SetSel 声明如下：

```
void SetSel(DWORD dwSeletion);
void SetSel(int nStartChar, int nEndchar);
```

参数 dwSelection 的低位字为起始位置，高位字为终止位置。参数 nStartChar 和 nEndChar 所定义的字符范围组成了新的选择文本。在这两个版本的 SetSel 中，如起始和终止位置分别为 0 和 -1，编辑控制中的全部文本都将被选中。

示　例：

```
int nStart = 10, nEnd = 20;
CEdit* pEdit;
// create control...
pEdit->SetSel(nStart, nEnd);
```

语　法

函数 ReplaceSel

函数 ReplaceSel **声明如下**:

```
void ReplaceSel(const char FAR * lpNewText);
```

参数 lpNewText **指向用来替换当前选中文本的新的选中文本。如果没有选中的文本,函数只是在当前插入点插入** lpNewText **指向的文本。**

示　例:

```
CEdit * pEdit;
// create control...
pEdit- > ReplaceSel("New text");
```

注　意　**可以利用函数** ReplaceSel **删除部分编辑控制文本,具体是先选择要删除的部分,然后再用一空串替换。**

16.3　下压按钮控制

下压按钮控制从心理上讲也许是功能最强的控制(从未听说过有核列表框和核选中框),从这种意义上讲,下压按钮代表了控制的基本观点——按击一下控制,就会发生一些事情。

新　词

命令按钮(command button)

下压按钮也称为命令按钮。

注　意　MFC **库中提供了** CButton **类,它的实例可创建下压按钮控制、分组框、选中框和圆按钮。这些控制按指定的按钮风格创建。许多从事** OOP **程序设计的程序员认为这种多态性是一种怪异的(糟糕的)设计。采用类** CButton **的目前这种形式是因为** MFC **库是** Windows API **函数的一个** OOP **外壳。**

这一课接下来主要集中讨论用类 CButton **处理下压按钮控制。第** 17 **天将介绍** CButton **中有关分组框、选中框和圆按钮等方面的内容。**

新　词

缺省按钮(default button)　　**非缺省按钮**(nondefault button)

有两种基本的下压按钮控制:缺省按钮和非缺省按钮,缺省按钮的边缘比非缺省按钮稍粗。在对话框中按 Enter **键相当于按击缺省按钮。在一个分组内只能有一个缺省按钮。可以用** Tab **键选择新的缺省按钮。但这些特性只有在对话框中才起作用。如果非对话框的窗口拥有下压按钮控制,它直观上只显示一个缺省按钮——但不支持其功能。**

CButton 类

类 CButton 为 CWnd 的派生类，它声明了很少几个成员函数。与下压按钮控制有关的 CButton 成员函数有：

- 成员函数 Create，它用类构造函数建立一个下压按钮实例。
- 成员函数 GetButtonStyle 和 SetButtonStyle，用它可以查询和设置下压按钮的风格。例如，用这两个函数可判断一个按钮是否是缺省按钮。

语　法

函数 Create

函数 Create 的声明如下：

```
BOOL Create(const char FAR* lpCaption,      // button label
            DWORD dwStyle,                  // style
            const RECT& rect,               // area
            CWnd* pParentWnd,               // parent Window
            UINT nID);                      // control ID
```

参数 lpCaption 指定按钮的标签或标题。参数 dwStyle 定义按钮的确切类型，此参数在类 CButton 的 Create 函数中扮演的角色比在其他类的 Create 函数中的要重要得多。下压按钮的典型风格为 WS_CHILD，WS_VISIBLE，BS_PUSHBTTON（或 BS_DEFPUSHBUTTON）和 WS_TABSTOP（当父窗口为个对话框时）。参数 rect 指定控制的区域和位置。参数 pParent 为指向属主窗口的指针，属主窗口可以是一个窗口，也可以是一对话框。参数 nID 指定唯一的控制 ID。

示　例：

```
CButton* pBtn;
dwBtnStyle = WS_CHILD ©¦WS_VISIBLE ©¦BS_PUSHBTTON;
pBtn->Create("Calc", dwBtnStyle, rect, NULL, ID_CALC_BTN);
```

语　法

函数 GetButtonStyle 和 SetButtonStyle

函数 GetButtonStyle 和 SetButtonStyle 声明如下：

```
UINT GetButtonStyle() const;
void SetButtonStyle(UINT nStyle, BOOL bRedraw = TRUE);
```

参数 nStyle 指定 CButton 实例的新风格，布尔型参数 bRedraw 指示是否重绘控制。

示　例：

```
UNIT nBtnStyle;
CButton* pBtn1;
CButton* pBtn2;
// create controls...
nBtnStyle = pBtn1->GetButtonStyle();
pBtn2->SetButtonStyle(nBtnStyle);
```

处理按钮消息

当按击一按钮时，控制将发送 BN_CLICKED 通知消息给其父窗口，父窗口将根据按钮的 ID，调用负责处理消息的成员函数来响应这条消息。例如，如果建立了一个 ID 为

ID_EXIT_BTN 的控制，本消息处理函数可为：

```
// other declarations
afx_msg void HandleExitBtn();
// other declarations
BEGIN_MESSAGE_LOOP(_CAppWindow_, CFrameWnd)
// other possible message mapping macros
ON_BN_CLICKED(ID_EXIT_BTN, HandleExitBtn)
// other possible message mapping macros
END_MESSAGE_LOOP()
```

本例示意了怎样用消息映射宏 ON_BN_CLICKED 将 ID_EXIT_BTN 通知消息映射到成员函数 HandleExitBtn。

操作按钮

利用函数 CWnd::EnableWindow 可以禁用、起用、显示和隐藏按钮。禁用的按钮具有褪色的灰色标题，它不会对鼠标的按击产生反应。使用函数 CWnd::EnableWindow 可以起用或禁用按钮。函数只接收一个简单的布尔型值，它指定起用(当取值为 TRUE)或禁用(当取值为 FALSE)按钮。可以这样调用函数 EnableWindow：

```
AButton_ > EnableWindow(FALSE); // disable button
AButton_ > EnableWindow(TRUE); // enable button
```

可使用无参数布尔型函数 CWnd::IsWindowEnabled 来查询一个按钮的起用状态。IsEnabledWindow 调用示范：

```
// toggle the enabled state of a button
if (AButton_ > IsWindowEnabled())
Abutton_ > EnableWindow(FALSE);      // disable button
else
AButton_ > EnableWindow(TRUE);       // enable button
```

同样，可以使用函数 CWnd::ShowWindow 来隐藏和显示按钮。此函数带一个参数值，当其取值为常量 WS_HIDE 时隐藏按钮，取值为 WS_SHOW 则显示按钮。布尔型函数 CWnd::IsWindowVisible 用于查询按钮的可见性，此函数不带参数。

函数 ShowWindow 和 IsWindowVisible 的调用示范：

```
// toggle the visibility of a button
if(AButton_ > IsWindowVisible())
        AButton_ > ShowWindow(WS_HIDE);   // hide button
else
        AButton_ > ShowWindow(WS_SHOW); // show button
```

命令行计算器

我们来看一个应用程序——命令行计算器应用程序(COCA)，该程序中使用了静态文本、单行编辑控制、多行编辑控制和下压按钮。这个较复杂的应用程序实现了浮点计算器，它用编辑控制代替了按钮。比起那种典型的按钮密集的计算器 Windows 应用程序，这种界面的直观性差一些。但是，这种界面可以支持更多的数学功能，而不需要新增加按钮。在 COCA 中，计算器由如下的控制组成：

- 表示第一和第二操作数的两个编辑控制接受整数、浮点数以及单字母变量名，即 A 到 Z。
- 一个表示运算符的编辑控制支持计算器的四种基本数学运算和乘方(用插入符^表示乘方)。
- 一个编辑控制显示数学运算的结果。
- 一个编辑控制显示出错信息。
- 一个多行编辑控制，用它可以把结果编辑控制中的数值存放到 26 个以 A 到 Z 命名的字母变量其中的一个当中。多行编辑显示这些变量的当前值，并允许查看和编辑这些数值。可以用竖直滚动条查看不同变量中的值。
- 多个静态文本控制可标记不同的编辑控制。特别要强调的是表示“Error Message”框的静态控制。如果点中相应的静态文本，“Error Message”的文本将被清除。
- 一个只有单个 Exit 项的菜单。
- 标题为“Calc”的下压按钮使用由“Operand”编辑控制指定的操作数执行“Operator”编辑控制中指定的操作。
- 标题为“Store”的下压按钮将结果编辑控制的内容保存到多行编辑控制中的当前选中行。
- 标题为“Exit”的下压按钮在得到确认后将终止应用程序。

程序支持“Store”按钮控制的如下特殊性能：

- 如应用程序企图执行一非法运算符，“Store”按钮将被禁用。这一特性示意了当产生了一定的条件(此例中，为特定的计算错误)时禁用按钮的情形。
- 当在“Error Message”静态文本上单击鼠标时，存储下压按钮将被起用。同样，当成功地进行了一次数学运算时，该按钮也会被起用。

计算器应用程序演示了如下的操作：

- 使用单行编辑控制进行简单输入；
- 使用多行编辑控制查看和编辑信息；
- 存取和编辑面向行的文本；
- 模拟响应鼠标按击的静态文本；
- 编写定制的 GetLine 和 GetLineLenght 成员函数；
- 扩充 CEdit 类以包容上述两个成员函数；
- 使用下压按钮；
- 禁用和起用按钮；
- 将加速键与下压按钮进行关联(这一特性允许程序中使用按键)。

在\MSVC\VC21DAY 下建立 CTLBTN1 子目录，并将所有的项目文件存在新目录下。本项目的.MAK 文件应包含文件 CTLBTN1.CPP，CTLBTN1.RC 和 CTLBTN1.DEF。最后一个文件与第 14 天中表 14.2 的 MINWINAP.DEF 相似。

首先编译并运行本应用程序，以便了解计算器是如何工作的。试试键入不同的数值操

作数和所支持的运算符，并击中“Calc”按钮。每次，结果出现在结构框中，并覆盖掉先前的结果。试试用 0 除一个数以试验错误处理功能。

使用单字母变量很方便，所有这些变量都已初始化为 0。第一步，先用它们存储非零值。执行一个运算操作，然后在“Variable”编辑框内单击鼠标。选择含有变量 A 的第一行。现在击中“Store”按钮（或按 Alt-S）就会看见“Result”框中的数值将出现在“Variable”编辑框中的第一行。变量名、冒号以及随后的空格符将重新出现在新的文本行。现在用变量 A 替换“Operand1”编辑框中的内容，再击中“Calc”按钮，“Result”编辑框将显示最后操作的结果。

清单 16.1 中列出了头文件 CTLBTN1.H 的源代码，其中定义了菜单项的命令常量和下压按钮的加速键。加速键 Alt-C，Alt-S 和 Alt-E 分别与各自的下压按钮 ID 相对应。清单 16.2 为 CTLBTN1.RC 资源描述文件，清单 16.3 为程序文件 CTLBTN1.CPP 的源代码。

键　入　**清单 16.1　头文件 CTLBTN1.H 的源代码**

```
1: # define CM_EXIT (WM_USER + 100)
2:
3: # define ID_CALC_BTN  107
4: # define ID_STORE_BTN 108
5: # define ID_EXIT_BTN  109
```

键　入　**清单 16.2　资源描述文件 CTLBTN1.RC**

```
1: # include < windows.h>
2: # include < afxres.h>
3: # include "ctlbtn1.h"
4:
5: BUTTONS ACCELERATORS
6: BEGIN
7:     "c", ID_CALC_BTN, ALT
8:     "s", ID_STORE_BTN, ALT
9:     "e", ID_EXIT_BTN, ALT
10: END
11:
12: EXITMENU MENU LOADONCALL MOVEABLE PURE DISCARDABLE
13: BEGIN
14:       MENUITEM "E&xit", CM_EXIT
15: END
```

键　入　**清单 16.3　程序文件 CTLBTN1.CPP 的源代码**

```
1: # include < stdlib.h>
2: # include < ctype.h>
3: # include < stdio.h>
4: # include < math.h>
5: # include < string.h>
6: # include < afxwin.h>
7: # include "ctlbtn1.h"
8:
9: // declare the constants that represent the sizes of the controls
10: const Wlbl = 100;
11: const Hlbl = 20;
12: const LblVertSpacing = 5;
13: const LblHorzSpacing = 40;
14: const Wbox = 100;
15: const Hbox = 30;
16: const BoxVertSpacing = 40;
17: const BoxHorzSpacing = 40;
18: const WLongbox = 4 * (Wbox + BoxHorzSpacing);
19: const Wvarbox = 2 * Wbox;
20: const Hvarbox = 3 * Hbox;
21: const Hbtn = 30;
22: const Wbtn = 80;
23: const BtnHorzSpacing = 30;
24: const MaxEditLen = 30;
25: const MAX_MEMREG = 26;
26:
27: // declare the ID_XXXX constants for the edit boxes
28: # define ID_OPERAND1_EDIT 101
29: # define ID_OPERATOR_EDIT 102
30: # define ID_OPERAND2_EDIT 103
31: # define ID_RESULT_EDIT 104
32: # define ID_ERRMSG_EDIT 105
33: # define ID_VARIABLE_EDIT 106
34:
35: class CxButton : public CButton
36: {
37: public:
38:
39:     BOOL Create(const char FAR* lpCaption, const RECT& rect,
40:     CWnd* pParentWnd, UINT nID, BOOL bIsDefault);
41:
42: };
43:
44: class CxEdit : public CEdit
45: {
46: public:
47:   // get an ASCIIZ string from a line
```

```
48:    BOOL GetLine(LPSTR lpString, int nStrSize, int nLineNumber);
49:    // get the line length given a line number
50:   int GetLineLength(int nLineNumber);
51:
52: };
53:
54: class CWindowApp : public CWinApp
55: {
56: public:
57: virtual BOOL InitInstance();
58: };
59:
60: // expand the functionality of CFrameWnd by deriving
61: // class CAppWindow
62: class CAppWindow : public CFrameWnd
63: {
64: public:
65:
66: CAppWindow();
67:
68: ~CAppWindow();
69:
70: protected:
71:    // declare the pointers to the various controls
72:    // first, the edit box controls
73:    CEdit*  Operand1Box;
74:    CEdit*  OperatorBox;
75:    CEdit*  Operand2Box;
76:    CEdit*  ResultBox;
77:    CEdit*  ErrMsgBox;
78:    CxEdit*  VariableBox;
79:    // then the static text controls
80:    CStatic*  Operand1Txt;
81:    CStatic*  OperatorTxt;
82:    CStatic*  Operand2Txt;
83:    CStatic*  ResultTxt;
84:    CStatic*  ErrMsgTxt;
85:    CStatic*  VariableTxt;
86:    // then the push buttons
87:    CxButton*  CalcBtn;
88:    CxButton*  StoreBtn;
89:    CxButton*  ExitBtn;
90:    // math error flag
91:    BOOL bInError;
92:    // coordinates for the Error Message static text area
93:    int nXulc, nYulc, nXlrc, nYlrc;
94:
95:    // handle clicking the left mouse button
96:    afx_msg void OnLButtonDown(UINT nFlags, CPoint point);
97:    // handle the calculation
98:    afx_msg void HandleCalcBtn();
99:    // handle the calculation
```

```
100:    afx- msg void CMCalc() { HandleCalcBtn();}
101:    // handle storing the result in a variable
102:    afx- msg void HandleStoreBtn();
103:    // handle storing the result in a variable
104:    afx- msg void CMStore() { HandleStoreBtn();}
105:    // handle exiting the application
106:    afx- msg void HandleExitBtn()
107:    { SendMessage(WM- CLOSE); }
108:    // handle exiting the application
109:    afx- msg void CMExit(){ SendMessage(WM- CLOSE); }
110:    // handle exiting the application
111:    afx- msg void OnExit(){ SendMessage(WM- CLOSE);}
112:    // enable a push button control
113:    virtual void EnableButton(CxButton* pBtn)
114:    { pBtn- > EnableWindow(TRUE); }
115:    // disable a push button control
116:    virtual void DisableButton(CxButton* pBtn)
117:    { pBtn- > EnableWindow(FALSE); }
118:    // handle creating the controls
119:    afx- msg int OnCreate(LPCREATESTRUCT lpCS);
120:    // handle keydown events
121:    afx- msg void OnKeyDown(UINT nChar, nRepCnt, UINT nFlags);
122:    // handle closing the window
123:    afx- msg void OnClose();
124:    // return a reference to member r based on individual
125:    // coordinates and dimenions
126:    void makeRect(int X, int Y, int W, int H, CRect& r);
127:    // obtain a number of a Variable edit box line
128:    double getVar(int lineNum);
129:    // store a number in the selected text of
130:    // the Variable edit box line
131:    void putVar(double nX);
132:
133:    // declare message map macro
134:    DECLARE- MESSAGE- MAP();
135: };
136:
137: BOOL CxButton::Create(const char FAR* lpCaption, const RECT& rect,
138:                       CWnd* pParentWnd, nUINT nID, BOOL bIsDefault)
139:
140: {
141:    DWORD dwBtnStyle = (bIsDefault = = TRUE) ?
142:                    BS- DEFPUSHBUTTON : BS- PUSHBUTTON;
143:
144:    return CButton::Create(lpCaption, WS- CHILD ©¦WS- VISIBLE ©¦
145:            WS- TABSTOP ©¦dwBtnStyle, rect, pParentWnd, nID);
146: }
147:
148: BOOL CxEdit::GetLine(LPSTR lpString, int nStrSize,
149:                      int nLineNumber)
150: {
151:    int nCopyCount;
```

```
152:    BOOL bResult;
153:
154:    if (nStrSize < = 0)
155:       return FALSE;
156:    bResult = (nStrSize > = GetLineLength(nLineNumber) + 1) ? TRUE
157:                                                            : FALSE;
158:   if (nStrSize = = 1)
159:   {
160:      lpString[0] = '\0';
161;      return bResult;
162:   }
163:   ((WORD FAR * )lpString)[0] = nStrSize;
164:   nCopyCount = (WORD)(SendMessage(EM_GETLINE, nLineNumber,
165:                                   long(lpString)));
166:   if (nCopyCount)
167:   {
168:      // Windows returns non-null-terminated string
169:      lpString[nCopyCount] = '\0';
170:      return bResult;
171:   }
172:   return FALSE;
173: }
174:
175: int CxEdit::GetLineLength(int nLineNumber)
176: {
177:    int nStartPos = -1;
178:
179:    if (nLineNumber > -1)
180:       nStartPos = LineIndex(nLineNumber);
181:    return (WORD) SendMessage(EM_LINELENGTH, nStartPos);
182: }
183:
184: CAppWindow::CAppWindow()
185: {
186:    // load accelerator resources
187:    LoadAccelTable("BUTTONS");
188:    // create the window
189:    Create(NULL,
190:       "Command-Oriented Caluclator Application (COCA)",
191:       WS_OVERLAPPEDWINDOW ©¦WS_MAXIMIZE,
192:          rectDefault, NULL, "EXITMENU");
193:       // clear the bInError flag
194:       bInError = FALSE;
195: }
196:
197: int CAppWindow::OnCreate(LPCREATESTRUCT lpCS)
198: {
199:    char s[81];
200:    char bigStr[6 * MAX_MEMREG + 1];
201:    char chC;
202:    int nX0 = 20;
203:    int nY0 = 30;
```

```
204:    int nX = nX0, nY = nY0;
205:    CRect r;
206:    DWORD dwStaticStyle = WS- CHILD ©¦WS- VISIBLE ©¦SS- LEFT;
207:    DWORD dwBoxStyle = WS- CHILD ©¦WS- VISIBLE ©¦WS- TABSTOP©¦
208:                       WS- BORDER ©¦ES- LEFT ©¦WS- AUTOHSCROLL ©¦
209:                       ES- UPPERCASE;
210:
211:    // create the first set of labels for the edit boxes
212:    makeRect(nX, nY, Wlbl, Hlbl, r);
213:    Operand1Txt = new CStatic();
214:    Operand1Txt- > Create("Operand1", dwStaticStyle, r, this);
215:    strcpy(s,"Operator");
216:    nX + = Wlbl + LblHorzSpacing;
217:    makeRect(nX, nY, Wlbl, Hlbl, r);
218:    OperatorTxt = new CStatic();
219:    OperatorTxt- > Create(s, dwStaticStyle, r, this);
220:    nX + = Wlbl + LblHorzSpacing;
221:    makeRect(nX, nY, Wlbl, Hlbl, r);
222:    Operand2Txt = new CStatic();
223:    Operand2Txt- > Create("Operand2", dwStaticStyle, r, this);
224:    nX + = Wlbl + LblHorzSpacing;
225:    makeRect(nX, nY, Wlbl, r);
226:    ResultTxt = new CStatic();
227:    ResultTxt- > Creat("Result", dwStaticStyle, r, this);
228:    // create the operand1, operator, operand2, and result
229:    // edit boxes
230:    nX = nX0;
231:    nY + = Hlbl + LblVertSpacing;
232:    makeRect(nX, nY, Wbox, Hbox, r);
233:    Operand1Box = new CEdit();
234:    Operand1Box- > Create(dwBoxStyle, r, this, ID- OPERAND1- EDIT);
235:    Operand1Box- > LimitText(); // set no limit for text
236:    nX + = Wbox + BoxHorzSpacing;
237:    makeRect(nX, nY, Wbox, Hbox, r);
238:    OperatorBox = new CEdit();
239:    OperatorBox- > Create(dwBoxStyle, r, this, ID- OPERATOR- EDIT);
240:    OperatorBox- > LimitText(); //set no limit for text
241:    nX + = Wbox + BoxHorzSpacing;
242:    makeRect(nX, nY, Wbox, Hbox, r);
243:    Operand2Box = new CEdit();
244:    Operand2Box- > Create(dwBoxStyle, r, this, ID- OPERAND2- EDIT);
245:    Operand2Box- > LimitText(); //set no limit for text
246:    nX + = Wbox + BoxHorzSpacing;
247:    makeRect(nX, nY, Wbox, Hbox, r);
248:    ResultBox = new CEdit();
249:    ResultBox- > Create(dwBoxStyle, r, this, ID- RESULT- EDIT);
250:    ResultBox- > LimitText(); // set no limit for text
251:    // create the static text and edit box for the error message
252:    nX = nX0;
253:    nY + = Hbox + BoxVertSpacing;
254:    // store the coordinates for the static text area
255:    nXulc = nX;
```

```
256:    nYulc = nY;
257:    nXlrc = nX + Wlbl;
258:    nYlrc = nY + Hlbl;
259:    makeRect(nX, nY, Wlbl, Hlbl, r);
260:    ErrMsgTxt = new CStatic();
261:    ErrMsgTxt-> Create("Error Message", dwStaticStyle, r, this);
262:    nY + = Hlbl + LblVertSpacing;
263:    makeRect(nX, nY, WLongbox, Hbox, r);
264:    ErrMsgBox = new CEdit();
265:    ErrMsgBox-> Create(dwBoxStyle, r, this, ID- ERRMSG- EDIT);
266:    ErrMsgBox-> LimitText(); // set no limit for text
267:    // create the static text and edit box for the single-letter
268:    // variable selection
269:    nY + = Hbox + BoxVertSpacing;
270:    makeRect(nX, nY, Wlbl, Hlbl, r);
271:    VariableTxt = new CStatic();
272:    VariableTxt-> Create("Variables", dwStaticStyle, r, this);
273:    nY + = Hlbl + LblVertSpacing;
274:    bigStr[0] = '\0';
275:    // build the initial contents of the Variable edit box
276:    for (chC = 'A'; chC < = 'Z'; chC+ + ) {
277:      sprintf(s, "%c: 0\r\n", chC);
278:      strcat(bigStr, s);
279:    }
280:    makeRect(nX, nY, Wvarbox, Hvarbox, r);
281:    VariableBox = new CxEdit();
282:    VariableBox- > Create(dwBoxStyle ©¦ES- MULTILINE ©¦WS- HSCROLL
283:                  ©¦WS- VSCROLL ©¦ES- AUTOVSCROLL,
284:                        r, this, ID- VARIABLE- EDIT);
285:    VariableBox- > LimitText(); // set no limit for text
286:    VariableBox-> SetWindowText(bigStr);
287:    // create the Calc push button
288:    nX + = Wvarbox + BtnHorzSpacing;
289:    CalcBtn = new CxButton();
290:    makeRect(nX, nY, Wbtn, Hbtn, r);
291:    CalcBtn- > Create("&Calc", r, this, ID- CALC- BTN, TRUE);
292:    // create the Store Btn
293:    nX + = Wbtn + BtnHorzSpacing;
294:    makeRect(nX, nY, Wbtn, Hbtn, r);
295:    StoreBtn = new CxButton();
296:    StoreBtn- > Create("&Store", r, this, ID- CALC- BTN, TRUE);
297:    // Create the Exit Btn
298:    nX + = Wbtn + BtnHorzSpacing;
299:    makeRect(nX, nY, Wbtn, Hbtn, r);
300:    ExitBtn = new CxButton();
301:    ExitBn-> Create("&Exit", r, this, ID- EXIT- BTN, FALSE);
302:    return CFrameWnd::OnCreate(lpCS);
303: }
304:
305: CAppWindow::～CAppWindow()
306: {
307:    // delete the controls
```

```
308:    delete Operand1Box;
309:    delete OperatorBox;
310:    delete Operand2Box;
311:    delete ResultBox;
312:    delete ErrMsgBox;
313:    delete VariableBox;
314:    delete Operand1Txt;
315:    delete OperatorTxt;
316:    delete Operand2Txt;
317:    delete ResultTxt;
318:    delete ErrMsgTxt;
319:    delete VariableTxt;
320:    delete CalcBtn;
321:    delete StoreBtn;
322:    delete ExitBtn;
323: }
324:
325: void CAppWindow::OnLButtonDown(UINT nFlags, CPoint point)
326: {
327:    int nX = point.x;
328:    int nY = point.y;
329:
330:    if (nX >= nXulc && nX <= nXlrc &&
331:        nY >= nYulc && nY <= nYlrc) {
332:        ErrMsgBox-> SetWindowText("");
333:        EnableButton(StoreBtn);
334:    }
335: }
336:
337: void CAppWindow::HandleCalcBtn;
338: {
339:    double x, y, z;
340:    char opStr[MaxEditLen + 1];
341:    char s[MaxEditLen + 1];
342:
343:    // obtain the string in the OPerand1 edit box
344:    Operand1Box- > GetWindowText(s, MaxEditLen);
345:    // does the Operand1Box contain the name
346:    // of a single- letter variable?
347:    if (isalpha(s[0]))
348:       // obtain value from the Variable edit control
349:       x = getVar(s[0] - 'A');
350:    else
351:       // convert the string in the edit box
352:       x = atof(s);
353:
354:    // obtain the string in the Operand2 edit box
355:    Operand2Box- > GetWindowText(s, MaxEditLen);
356:    // does the Operand2Box contain the name
357:    // of a single- letter variable?
358:    if (isalpha(s[0]))
359:       // obtain value from the Variable edit control
```

```
360:      y = getVar(s[0] - 'A');
361:    else
362:      // convert the string in the edit box
363:      y = atof(s);
364:
365:    // obtain the string in the Operator edit box
366:    OperatorBox->GetWindowText(opStr, MaxEditLen);
367:    // clear the error message box
368:    ErrMsgBox->SetWindowText("");
369:    bInError = FALSE;
370:    // determine the requested operation
371:    if (strcmp(opStr, "+") == 0
372:      z = x + y;
373:    else if (strcmp(opStr, "-") == 0)
374:      z = x - y;
375:    else if (strcmp(opStr,"*") == 0)
376:      z = x * y;
377:    else if (strcmp(opStr, "/") == 0)
378:      if (y != 0)
379:        z = x / y;
380:      else {
381:        z = 0;
382:        bInError = TRUE;
383:        ErrMsgBox->SetWindowText("Division by zero error");
384:      }
385:    }
386:    else if (strcmp(opStr, "^") == 0) {
387:      if(x > 0)
388:        z = exp(y * log(x));
389:      else {
390:        bInError = TRUE;
391:        ErrMsgBox->SetWindowText(
392:          "Cannot raise the power of a negative number");
393:      }
394:    }
395:    else {
396:      bInError = TRUE;
397:      ErrMsgBox->SetWindowText("Invalid operator");
398:    }
399:    // display the result if no error has occurred
400:    if (!bInError) {
401:      sprintf(s, "%g", z);
402:      ResultBox->SetWindowText(s);
403:      // enable the Store button
404:      EnableButton(StoreBtn);
405:    }
406:    else
407:      // disable the Store button
408:      DisableButton(StoreBtn);
409:    }
410:
411: void CAppWindow::HandleStoreBtn()
```

```
412: {
413:    char result[MaxEditLen + 1];
414:
415:    // get the string in the Result edit box
416:    ResultBox-> GetWindowText(result, MaxEditLen);
417:    // store the result in the selected text of
418:    // the Variable edit box
419:    putVar(atof(result));
420: }
421:
422: void CAppWindow::makeRect(int nX, int nY, int nW,
423:                           int nH, CRect& r)
424: {
425:    r.top = nY;
426:    r.left = nX;
427:    r.bottom = nY + nH;
428:    r.right = nX + nW;
429: }
430:
431: double CAppWindow::getVar(int nLineNum)
432: {
433:    int nLineSize;
434:    char s[MaxEditLen + 1];
435:
436:    if (nLineNum > = MAX_MEMREG) return 0;
437:    // get the size of the target line
438:    nLineSize = VariableBox- > GetLineLength(nLineNum);
439:    // get the line
440:    VariableBox- > GetLine(s, nLineSize, nLineNum);
441:    // delete the first three characters
442:    strcpy(s, (s+ 3));
443:    // return the number stored in the target line
444:    return atof(s);
445: }
446:
447: void CAppWindow::putVar(double x)
448: {
449:    DWORD dwSelPos;
450:    WORD wStartPos, wEndPos;
451:    int nLineNum;
452:    int nLineSize;
453:    char s[MaxEditLen + 1];
454:
455:    // locate the character position of the cursor
456:    dwSelPos = VariableBox- > GetSel();
457:    wStartPos = LOWORD(dwSelPos);
458:    wEndPos = HIWORD(dwSelPos);
459:    // turn off the selected text
460:    if (wStartPos ! = wEndPos) {
461:       dwSelPos = MAKELONG(wStartPos, wStartPos);
462:       VariableBox- > SetSel(dwSelPos);
463:    }
```

```
464:    // get the line number where the cursor is located
465:    nLineNum = VariableBox->LineFromChar(-1);
466:    // get the line size of line lineNum
467:    nLineSize = VariableBox->GetLineLength(nLineNum);
468:    // obtain the text of line lineNum
469:    VariableBox->GetLine(s, nLineSize, nLineNum);
470:    // build the new text line
471:    sprintf(s,"%c: %g", s[0], x);
472:    // get the character positions for the deleted line
473:    wStartPos = (WORD) (VariableBox->LineIndex(-1));
474:    wEndPos = (WORD) (wStartPos + VariableBox->LineLength(-1));
475:    // select the currrent line
476:    dwSelPos = MAKELONG(wStartPos, wEndPos);
477:    // replace the current line with the new line
479:    VariableBox->ReplaceSel(s);
480: }
481:
482: void CAppWindow::OnKeyDown(UINT nChar, UINT nRepCnt, UINT nFlags)
483: {
484:    if (nFlags & 0x2000 == 1)
485:       switch (nChar) {
486:          case 'c':
487:             HandleCalcBtn();
488:             break;
489:          case 's':
490:             HandleStoreBtn();
491:             break;
492:          case 'e':
493:             HandleExitBtn();
494:             break;
495:          default:
496:             MessageBeep(0);
497:       }
498: }
499:
500: void CAppWindow::OnClose()
501: {
502:   if (MessageBox("Want to close this application",
503:         "Query", MB_YESNO ¦ MB_ICONQUESTION) == IDYES)
504:      DestroyWindow();
505: }
506:
507: BEGIN_MESSAGE_MAP(CAppWindow, cFrameWnd)
508:       ON_WM_LBUTTONDOWN()
509:       ON_BN_CLICKED(ID_CALC_BTN, HandleCalcBtn)
510:       ON_COMMAND(ID_CALC_BTN, CMCalc)
511:       ON_BN_CLICKED(ID_STORE_BTN, HandleStoreBtn)
512:       ON_COMMAND(ID_STORE_BTN, CMStore)
513:       ON_BN_CLICKED(ID_EXIT_BTN, HandleExitBtn)
514:       ON_COMMAND(ID_EXIT_BTN, CMExit)
515:       ON_COMMAND(CM_EXIT, OnExit)
516:       ON_WM_CREATE()
```

```
517:       ON_WM_KEYDOWN()
518:       ON_WM_CLOSE()
519: END_MESSAGE_MAP()
520:
521: // Construct the CWindowApp's m_pMainWnd data member
522: BOOL CWindowApp::InitInstance()
523: {
524:    m_pMainWnd = new CAppWindow();
525:    m_pMainWnd->ShowWindow(m_nCmdShow);
526:    m_pMainWnd->UpdateWindow();
527:    return TRUE;
528: }
529:
530: // application's constructor initializes and runs the app
531: CWindowApp WindowApp;
```

图 16.1 中显示了程序 CTLBTN1.EXE 的一个运行范例。

图 16.1　程序 CTLBTN1.EXE 的一个运行范例

分　析

清单 16.3 中的程序在第 10 到 33 行声明了三组常量。第一组位于第 10 到 23 行，声明了高度、宽度、垂直间距和水平间距常量，用它们可以确定各个控制的尺寸；第二组常量位于第 24 到 25 行，声明了编辑控制中的最大字符数和多行编辑控制中的变量个数；第三组常量位于第 28 到 33 行，声明了建立各编辑控制实例的 ID_xxxx 值。下压按钮控制 ID_xxxx 值这在头文件 CTLBTN1.H 中声明的，该文件已包含在本程序中。

COCA 应用程序声明了应用程序类 CWindowApp、窗口类 CAppWindow、扩展按钮类 CxButton 和扩展编辑框类 CxEdit。

> **注　意**　程序定义了类 CxButton，其中声明了函数 Create 的一个更专门的版本。此版本用典型的下压按钮风格建立正常的或缺省的下压按钮。用了这些风格，在客户窗口或对话框中就比较容易建立下压按钮控制。

类 CxEdit 是 CEdit 的子孙类，其中声明了成员函数 GetLine 和 GetLineLength。函数 GetLine 以 ASCIIZ 串为变元工作。函数声明如下：

```
BOOL GetLine (LPSTR lpString, int nStrSize, int nLineNumber);
```

lpString 为指向 ASCIIZ 串的指针，参数 nStrSize 指定至多拷贝多少字符数。nLineNumber 指明目标行号。函数在成功时返回 TRUE，否则为 FALSE。函数成员 GetLineLength 返回指定行的长度，函数 GetLineLength 声明为：

```
int GetLineLength(int nLineNumber);
```

参数 nLineNumber 指定要查找其长度的行的行号。

窗口类 CAppWindow 是静态文本和编辑控制的属主，类中声明了若干数据成员和成员函数。

CAppWindow 包含有不同数据成员组：

- 指向各种 CStatic，CEdit，CxEdit 和 CxButton 实例的指针（每个指针访问程序中出现的一个编辑控制）。
- 布尔型数据成员 bInError 可指示错误。
- 数据成员 nXulc，nYulc，nXlrc 以及 nYlrc，保存着包含“Error Message”静态文本的方框的坐标，（响应鼠标按击的成员函数 OnLButtonDown 检查鼠标是否击中矩型框内部，如果是，将清除“Error Message”编辑控制文本）。

类 CAppWindow 中包含一个构造函数、一个析构函数以及若干处理鼠标按击和响应菜单选择的消息响应成员函数。

窗口类构造函数执行如下操作：

- 调用 LoadAccelTable 函数并指定加速键资源为 BUTTONS，加载菜单资源。
- 调用 Create 成员函数建立窗口实例。
- 将数据成员 bInError 赋值为 FALSE。

调用函数 Create 时指定了窗口标题，WS_OVERLAPPEDWINDOW 和 WS_MAXIMIZE 风格以及 EXITMENU 菜单资源。WS_MAXIMIZE 风格产生一个最大化窗口。调用函数 Create 还可建立静态文本、编辑框和下压按钮的实例。

成员函数 EnableButton 和 DisableButton 通过调用 CWnd::EnableWindow 函数分别起用和禁用某下压按钮。此函数由其他函数以 StoreBtn 为参数值调用。

成员函数 makeRect 产生控制所占的矩形域的有关数值，具体如下：

- 矩形域的左上角的坐标
- 矩形的宽度

· 矩形的高度

成员函数 OnCreate 响应 WS- CREATE 消息并建立各种附属于窗口的控制。函数执行如下的任务:

· 调用 CStatic 构造函数建立静态文本控制以标记操作数、运算符和结果编辑控制。
· 建立操作数、运算符和结果编辑框。
· 建立 CxButton 实例,每个实例具有唯一的 ID 和标题。
· 计算包含“Error Message”静态文本的矩形框的左上角和右下角的坐标,并存在数据成员 nXulc, nYulc, nXlrc 和 nYlrc 中。
· 建立错误消息静态文本控制和编辑控制。
· 利用串变量 bigStr 产生“Variable”框的内容,建立变量多行编辑控制。
· 返回 CFrameWnd::OnCreate(LpCS) 表达式的值。

成员函数 OnCreate 在建立静态文本控制时使用 WS- CHILD©WS- VISIBLE©$S-LEFT 风格。建立控制时还涉及到私有成员函数 makeRect,它产生所需要的 CRect 对象。局部变量 nX 增加(Wlbl+ LblHorzSpacing),作为下一个静态文本控制的 X 坐标。使用这种方法可系统地增加控制区域的 X 和/或 Y 坐标。

操作数、运算符和结果编辑框的实例,可以通过数据成员 Operand1Box, OperatorBox, Operand2Box 和 ResultBox 分别进行存取。每一个 CEdit 实例建立时都有自己的 ID- xxxx 常量和一个空编辑框。编辑框大小都是一样的。编辑控制的风格包括 ES- UPPERCASE 风格,由于此风格的原因,在这些编辑控制中键入的单字母变量将自动转换为大写。

CxButton 实例的标题使用“&”字符给热键加下划线。使 CxButton::Create 的最后一参数取值为 TRUE,这样建立作为缺省控制的第一个下压按钮。使函数 Create 最后一个参数值为 FALSE 建立作为正常控制的另两个按钮。接着 Calc 按钮作为缺省下压按钮出现。但是,外表可能是个骗局—— 由于 Calc 按钮的属主为一窗口而非对话框,因而它缺少缺省按钮的功能。

“Variable”编辑控制在建立时,附加有 ES- MULTILINE, WS- HSCROLL, WS-VSCROLL 和 ES- AUTOVSCOLL 风格,这些风格使一个编辑控制具有全功能的垂直和水平滚动条,通过调用函数 SetWindowText,函数 OnCreate 将 bigStr 的内容写到编辑控制中。

成员函数 OnLButtonDown 完成的任务很简单,它检查鼠标按击是否发生在错误消息静态文本控制的矩形框中,如果是,函数将执行以下操作:

· 调用函数 SetWindowText 清除错误消息框。
· 调用函数 EnableButton 起用“Store”按钮。

成员函数 HandleCalcBtn 响应“Calc”按钮发出的通知消息,相应的消息映射宏使用 ID- CALCBTN 的消息 ID,即 Calc 按钮的 ID,建立“Calc”按钮实例时也要用到它。该函数完成以下操作:

· 从 Operand1 编辑框中获得第一个操作数。

- 用与第一个相同的方式获取第二个操作数(函数将实际的(数值)第二操作数保存到变量 y 中)。
- 将“Operator”编辑框中的文本复制到局域变量 opStr 中。
- 清除错误消息文本框并将数据成员 bInError 赋值为 FALSE。
- 用一系列的 if 和 if-else 语句判断所需操作——所支持的运算符有+，-，*，/，和^ (乘方)——并且如果函数检测到错误，置数据成员 bInError 为 TRUE，并在错误消息框中显示消息。
- 如数据成员 bInError 为 FALSE，显示“Result”框中的结果(先把结果由 double 转换成字符串，然后调用 SetWindowText 函数写入结果框)。随后起用“Store”按钮(如成员 bInError 为 TRUE，调用成员函数 DisableButton 将禁止“Store”下压按钮)。

当成员函数 HandleCalcBtn 从“Operand1”编辑框获得第一操作数时，控制可能包括单字母变量(A 到 Z)的名字或浮点数。函数利用 GetWindowText 将编辑控制文本的一个副本保存到局部变量 s 中。函数检查变量 s 的第一个字符，如果字符是一个字母，第一操作数为单字母变量。随后，函数调用保护的成员函数 getVar 获取该变量的有关值。如第一字符不是字母，函数使用 atof 将变量 s 的内容转换成 double 型值。在这两种情况中，函数都将实际的(数值)第一操作数保存到变量 x 中。

成员函数 CMCalcBtn 处理由 Alt-C 加速键发出的命令消息，相应消息 ID 宏指定为 ID_CALC_BTN，在资源文件中，此 ID 与 Alt-C 相关联(也是“Calc”按钮的 ID)。函数 CMCalcBtn 调用成员函数 HandleCalcBtn 执行要求的计算。

成员函数 HandleStoreBtn 将“Result”框中的内容保存到单字母变量中，函数首先调用 GetWindowText 从“Result”编辑框中获得字符串，随后再调用保护成员函数 putVar 将结果字符串实际保存到变量编辑框的当前插入点。

成员函数 CMStore 截取 Alt-S 热键，产生 ID 值为 ID_STORE_BTN 的命令消息。函数 CMStore 很少调用成员函数 HandleStoreBtn.

成员函数 getVar 返回“Variable”编辑框中存在行号为 nlineNum 的行中的数值，本函数完成以下任务:

- 当 nlineNum 大于或等于常量 MAX_MEMREG 时，退出并返回 0 值。
- 调用 GetLineLength(nLineNum)获得目标行的大小。
- 调用函数 GetLine，检索行号为 nLineNum 的字符串。
- 删除检索行的头三个字符(这一步应将带数字的字符串存在目标行中)。
- 以 s 为变元调用函数 atof，返回 double 型值。

成员函数 putVar 将“Result”框中的数值保存到一个变量中，该变量位于包含有文本插入位置的行上，本函数完成以下任务:

- 通过调用 GetSel 函数，定位当前字符光标位置，该函数用局部变量 dwSelPos 返回字符的起始和终止位置，并利用预定义宏 HIWORD 和 LOWORD 在局部变量 wStartPos 和 wEndPos 中保存起始和终止位置。

- 关闭任一选中的文本，比较 wStartPos 和 wEndPos 的值。如不一致，计算 dwSelPos 的新值，使高位字和低位字都为 wStartPos（函数然后以 dwSelPos 为参数调用 SetSel 以关闭选中的文本）。
- 调用函数 GetLineFromChar 获得光标所在行的行号。
- 调用 GetLineLength 函数获得目标行长度。
- 调用 GetLine 函数检索目标行中的文本。
- 建立新的文本行。
- 获得要替换的行的起始和终止字符位置。
- 调用 SetSel 函数选中将被替换的行。
- 用一新行替换当前行。

成员函数 HandleExitBtn 响应“Exit”按钮的通知消息。该函数发送 WM_CLOSE 消息给父窗口，成员函数截获 Alt-E 热键，产生一条其 ID 值为 ID_EXIT_BTN 的命令消息。函数 CMExit 也发送 WM_CLOSE 消息给父窗口。

小　　结

今天这一课主要讲述了静态文本、编辑框和下压按钮控制。使用这些控制和其他控制使 Windows 应用程序更加生动、用户界面更始终如一。今天这一课的主要内容有：

- 建立静态文本控制并在运行时对其文本进行操作。
- 单行和多行编辑框控制在输入对话框中键入文本并对其进行编辑。
- 两类下压按钮控制——缺省和非缺省——可收到立竿见影的效果。

在第 17 天中，我将介绍分组控制：组的类，选中框和图形控制，这些控制用来调整某些任务的执行，诸如在文本编辑器中搜索和替换文本。

问　与　答

问：怎样建立多行静态文本控制的字符串。

答：建立一个多行字符串，如"This is\na multiline"（注意嵌入的\n 分行符），并传递给成员函数 CStatic::Create，作为它的第 1 个变元。

问：选择正确的控制风格是否重要？

答：非常重要。风格可精细调整控制的外观、任选部件和行为。

问：为什么程序 CTLBTN1.CPP 用局域变量如 nX 和 nY 确定一个控制的位置？为什么不要数值常数代替？

答：使用变量，如 nX 和 nY，可以指定一组控制相对于另一组控制的位置。用这种方法能非常方便地移动控制。相反地，使用数值常量就确定了一个控制的绝对位置，在这种情况下，移动控制时将拖出一连串新数值。

作　　业

作业部分所提供的测验题有助于巩固理解所学内容,通过练习可以积累运用所学知识的经验。要尽量弄懂测验和练习的答案后再继续下一天的课程。附录 B 的“答案”中提供了解答。

测　验

1. 对还是错?
 所有静态文本控制的文本都没法改变。
2. 对还是错?
 SS_CENTER 使多行静态文本控制的每行都居中。
3. 对还是错?
 只有当程序需要设置或查询控制中的文本时,静态文本控制才需要一相应的指针访问它。
4. 对还是错?
 每个编辑控制都需有一个相应的指针访问它。
5. 对还是错?
 API Windows 函数 EnableWindow 和 DisableWindow 可以禁止任何控制。
6. 对还是错?
 可以用 ON_COMMAND 对下压按钮发出的 Windows 消息进行映射。

练　习

1. 在程序 CTLBTN1.CPP 中增加三角函数、反三角函数、双曲线函数和反双曲线函数的功能。
2. 在程序 CTLBTN1.CPP 的一个副本中,试试改变确定各种控制的尺寸和间距的常量值。

第17天

分组控制

Windows 支持以选中框和圆按钮作为软件开关。这些控制出现在典型的搜索和替换对话框中,影响着文本搜索或替换的范围、方向以及是否区分大小写等。今天这一课,我们将学习以下内容:

- 选中框控制
- 圆按钮控制
- 分组控制

今天的课程还将演示如何响应这些控制所发出的消息,以及如何使用 ForEach 迭代算符来操作选中框和圆按钮控制。

新　词

分组框控制

分组框控制是按选中框和圆按钮控制的外观和逻辑分组的特殊控制。

17.1　选中框控制

选中框控制是切换选中标记的特殊按钮。该控制的实例一般以小矩形框按钮和一个标题的形式出现。缺省时,标题在方框的右边。击中小方框则切换控制的选中标记。可以把选中框当作一个可以清除和设置的二进制位。选中框的实例可出现在组合框的内部或外部,相互不排斥——切换选中框不会影响其他选中框的选中状态。

注　意　把选中框放在组内(对话框中)有两个目的:首先,组合框提供了直观的分组,从而可使应用程序用户明白选中框的目的;其次,可以连续检查组内选中框发出的通知消息,检测选中框的选中状态有无变化。

新　词

变灰(grayed)

Windows 中用三种状态描述一个选中框:选中、未选中和变灰。第 5 天的表 5.1 中列出了选中框控制的这些风格。变灰状态用灰色填充控制的矩形按钮。处于上述第三种状态时,该选中框控制为未确定(或叫“不理会”)状态。

CButton 类和选中框

Microsoft 基本类(MFC)提供了 CButton 类,它是 CWnd 的子孙类,此类提供选中框

控制以及其他控制的实例。第 16 天的课程中介绍了 CButton 类,讨论此类中有关下压按钮控制的内容。

本节我们将集中讨论类 CButton 中与选中框控制有关的内容。当然,选中框实例需要类构造函数和成员函数 Create 才能实现。表 17.1 中的选中框风格表明,有两种方式可以管理选中框控制的选中状态:自动和非自动(或称为手动)。在自动方式(由 BS-AUTOCHECK 和 BS- AUTO3STATE 风格指定)下,当击中控制时,Windows 切换选中状态。在手动方式下,由应用程序代码负责管理选中框的选中状态。

表 17.1　选中框控制风格

风　格	含　义
BS- CHECKBOX	指定一个矩形按钮右边有标题的选中框。
BS- AUTOCHECKBOX	和 BS- CHECKBOX 一样,但击中它时按钮将自动切换。
BS- 3STATE	和 BS- CHECKBOX 一样,但它的控制有三种状态:选中、未选中和变灰。
BS- AUTO3STATE	和 BS- 3STATE 一样,但选中它时按钮可自动切换。
BS- LEFTTEXT	将控制的标题设置在按钮的左边。

类 CButton 提供了用于设置和查询选中框状态的成员函数。成员函数 GetCheck 返回选中框的状态,它声明如下:

```
int GetCheck() const;
```

函数返回的是代表选中状态的 int 型值,值 0 表示控制未选中;值 1 表示控制被选中;值 2 表明控制处于未确定状态。值 2 只有在具有 BS- 3STATE 和 BS- AUTO3STATE 风格时才有效。

> **注　意**　为增强代码的可读性,建议你声明类似 BF- UNCHECKED, BF-CHECKED 和 BF- GRAYED 一类的常量来表示选中框控制的三种状态。

成员函数 SetCheck 可用来设置选中框控制的选中状态。函数 SetChect 声明如下:

```
void SetCheck(int nCheck);
```

参数 nCheck 指定选中框控制的新状态。

响应选中框消息

因为选中框是带有 BS- CHECKBOX 或 BS- AUTOCHECKBOX 风格的 CButton 实例,MFC 应用程序响应选中框发出的消息的方式与下压按钮相似,BN- CLICKED 宏产生选中框控制发出的消息和相应该消息的成员函数之间的映射。

17.2　圆按钮控制

圆按钮控制一般允许从两个或多个选项中选中一项。这种类型的控制具有圆形按

钮,且缺省情况下标题出现在按钮右边。选中一个圆按钮时,一个小实心圆就会出现在圆形按钮中。圆按钮需要按直观和逻辑组合放入组合框中。每组圆按钮只能有一个被选中,因而圆按钮是相互排斥的。

类 CButton 和圆按钮

MFC 应用程序用具有 BS_RADIOBUTTON 或 BS_AUTORADIOBUTTON 风格的 CButton 类建立圆按钮控制。表 17.2 中列出了圆按钮的风格。构造函数用 BS_AUTORADIOBUTTON 风格建立一个圆按钮。建立这些控制要用到类的构造函数和成员函数 Create。像选中框控制一样,圆按钮用成员函数 GetCheck 和 SetCheck 来查询和设置状态。与选中框不一样的是,圆按钮只有两种状态:选中和未选中。

和选中框控制一样,圆按钮控制发送相同类型的通知消息给它们的父窗口,处理圆按钮消息与处理选中框和下压按钮控制消息是一样的。

表 17.2　圆按钮控制的风格

风　格	含　义
BS_RADIOBUTTON	指定标题在右边的圆按钮
BS_AUTORADIOBUTTON	和 BS_RADIOBUTTON 一样,但击中它时按钮将自动切换
BS_LEFTTEXT	将控制标题设置在按钮左边

17.3　分 组 控 制

分组框控制包括圆按钮和选中框,分组框完成如下任务:

- 把圆按钮和/或选中框直观地分组,使这些控制之间的关系相对于应用程序使用者来说更加清晰。
- 把多个圆按钮逻辑分组。当选中一个圆按钮,同一组的其他控制自动清除选中状态。

新　词

包容控制(container control)
分组框控制是一种被称为包容控制的特殊类型控制。

编写应用程序时可以让组合框内的控制将控制状态已经改变的消息通知其父窗口。

类 CButton 与分组框

在 MFC 应用程序中,当建立的 CButton 实例指定为 BS_GROUPBOX 风格时,便可以建立一个分组框。与选中框,圆按钮和下压按钮控制不一样,分组框控制没有状态,另外,分组框控制不发送消息。

17.4　更新的计算器应用程序

第 16 天所提供的计算器应用程序供的运算功能可以扩展,这里要使其包括三角函

数。使用三角函数通常需要提供角度制的选择功能,即选择弧度、度或公制度(100 公制度等于 90 度)。用一个包含角度制圆按钮的分组框,便可以演示这些控制操作。加入选中框以用于调整计算器的其他操作,在这节的后面将会有详细叙述。

在\MSVC\VC21DAY 下建立子目录 CTLGRP1,将本项目所有文件存在新的目录下,.MAK 文件应包含文件 CTLGRP1.CPP,CTLGRP1.RC 和 CTLGRP1.DEF,最后一个文件与第 14 天中清单 14.2 的 MINWINAP.DEF 文件相似。

> **注　意**　这里我们用完整的应用程序来演示圆按钮、选中框和编辑文本控制的完整特性。虽然清单 17.3 中的程序很长,但你可以这样建立它:先拷贝程序 CTLBTN1.CPP,然后在其中加入新的函数和语句。

关于演示

更新的计算器应用程序(COCA Version2)演示了如下几个方面的内容:

- 选中框控制的基本使用。
- 圆按钮的基本使用。
- 响应圆按钮通知消息。
- 在分组框控制中回应选中的圆按钮控制的变化。
- 使用选中框控制改变消息响应函数的动作(该函数处理分组框,以响应对 一个新圆按钮的选择)。
- 使用迭代算子 ForEach 操作多个选中框和编辑框。
- 初始化选中框选中状态。
- 初始化圆按钮选中状态。

新版本的计算器应用程序包含如下控制:

- 标名为 Operand1, Operator, Operand2, Result 和 Error Message 的单行编辑控制。
- 标名为 Variables 的多行编辑控制。
- 用于标名这些编辑控制的静态文本控制。
- Calc, Store 和 Exit 下压按钮。
- Angle Mode 分组框包含 Radians, Degrees 和 Gradians 圆按钮。
- Substitute(替换)变量名的选中框可用相应的值来替换出现在操作数编辑框的单字母变量的名字(击中 Calc 下压按钮后,替换发生)。
- Echo Group 框的消息选中框在另一圆按钮被选中时允许组的通知消息处理程序显示一个消息框。
- 包含 Exit 和 Clear 主菜单项的菜单。

Clear 弹出式菜单有 Edit Controls 和 Check Boxes 两个菜单项。菜单项 Edit Controls 清除编辑控制的内容,但 Variable 编辑框除外。菜单项 Check Boxes 清除三个选中框的选中状态。

应用程序试验

清单 17.1 中列出了头文件 CTLGRP1.H 的源代码，清单 17.2 为资源描述文件 CTLGRP1.RC。资源文件中包括加速键表和菜单资源。清单 17.3 为程序文件 CTLGRP1.CPP 的源代码。程序包含# include 语句、常量、迭代函数以及应用程序类、类窗口及其控制的声明。

编译并运行计算器应用程序，先熟悉圆按钮、选中框和分组框支持的特性。当应用程序窗口出现后，使其最大化，以便看到全貌。Angle Mode 分组框显示 Degrees 圆按钮的初始选择。这一版本的应用程序支持 SIN，COS，TAN，ASIN，ACOS 和 ATAN 三角函数，可以在 Operator 编辑框中输入这些函数。在 Operand1 编辑框中输入数值 45，在 Operator 编辑框中键入 TAN。单击 Calc 按钮，就可以在 Result 编辑框中获得 45 的正切值(为 1)。通过单击 Store 按钮，把结果保存在变量 A 中。现在在 Operand1 编辑框中用字符# 替换数字 45，然后在 Operator 框的 TAN 字串前插入 A，单击 Calc 按钮。# 字符被替换成在 Result 编辑控制中的原选数值 1。Result 编辑控制现在显示 45，即 1 的反正切。使用# 字符是程序中新增的特性—— 在操作域编辑中键入字符# ，能够利用上次计算的结果。接下来，再次单击 Store 按钮。应用程序把数值 45 保存到变量 A 中，覆盖掉先前的值。到目前为止，我们已把 Result 编辑框的数存入多个变量中。

选中 Echo Group 消息框的选中框，然后单击任一圆按钮，应用程序显示一个有关不同角度之间转换信息的消息框，选中不同的圆按钮将产生不同的消息。这些消息响应 Angle Mode 组的通知消息。

选用不同的角度制来进行试验。每次在 Operand1 编辑框中输入一个值和一个三角函数，观察相同的语句在不同的角度制情况下结果有何不同。应注意，反正弦和反余弦函数可接受的值在- 1 与 1 之间。应用程序检测到这两个函数的非法参数值时，会显示出错消息。

最后，利用 Clear©Edit Control 和 Clear©Check Boxes 菜单项清除编辑框(Variables 编辑框除外)和选中框。

此程序中忽略了这样的特性，即单击 Error Message 标记时，清除出错信息和起用 Store 按钮。因为这样程序可以简短一些。

键　入　　清单 17.1　头文件 CTLGRP1.H 的源代码

```
1: # define CM_EXIT          (WM_USER + 100)
2: # define CM_CLEARBOX      (WM_USER + 101)
3: # define CM_CLEARCHK      (WM_USER + 102)
4:
5: # define ID_CALC_BTN      107
6: # define ID_STORE_BTN     108
7: # define ID_EXIT_BTN      109
```

键　入　**清单 17.2　资源描述文件 CTLGRP1.RC**

```
# include < windows.h>
# include < afxres.h>
# include "ctlgrp1.h"

BUTTONS ACCELERATORS
BEGIN
    "c", TD_CALC_BTN, ALT
    "s", ID_STORE_BTN, ALT
    "e", ID_EXIT_BTN, ALT
END

EXITMENU MENU LOADONCALL MOVEABLE PURE DISCARDABLE
BEGIN
    MENUITEM "E&xit", CM_EXIT
    POPUP "C&lear"
    BEGIN
        MENUITEM "&Edit Controls", CM_CLEARBOX
        MENUITEM "&Check Boxes", CM_CLEARCHK
    END
END
```

键　入　**清单 17.3　程序文件 CTLGRP1.CPP 的源代码**

```
# include < stdlib.h>
# include < ctype.h>
# include < stdio.h>
# include < math.h>
# include < string.h>
# include < afxwin.h>
# include "ctlgrp1.h"

// declare the constants that represent the sizes of the controls
const Wlbl = 100;
const Hlbl = 20;
const LblVertSpacing = 5;
const LblHorzSpacing = 40;
const Wbox = 100;
const Hbox = 30;
const BoxVertSpacing = 10;
const BoxHorzSpacing = 40;
const WLongbox = 4 * (Wbox + BoxHorzSpacing);
const Wvarbox = 2 * Wbox;
const Hvarbox = 3 * Hbox;
const Hbtn = 30;
const Wbtn = 80;
```

```
23: const BtnHorzSpacing = 30;
24: const BtnVertSpacing = 10;
25: const Hgrp = 150;
26: const Wgrp = 180;
27: const GrpHorzSpacing = 30;
28: const GrpVertSpacing = 10;
29: const Hchk = 30;
30: const Wchk = 200;
31: const ChkHorzSpacing = 30;
32: const ChkVertSpacing = 10;
33: const Hrbt = 30;
34: const Wrbt = 80;
35: const RbtHorzSpacing = 30;
36: const RbtVertSpacing = 30;
37: const RbtLeftMargin = 30;
38: // non-control related constants
39: const MaxEditLen = 30;
40: const MAX_MEMREG = 26;
41  // declare the ID_XXXX constants for the various controls
42: // (except the push buttons)
43: # define ID_OPERAND1_EDIT 101
44: # define ID_OPERATOR_EDIT 102
45: # define ID_OPERAND2_EDIT 103
46: # define ID_RESULT_EDIT 104
47: # define ID_ERRMSG_EDIT 105
48: # define ID_VARIABLE_EDIT 106
49: # define ID_SUBST_CHK 110
50: # define ID_ECHO_CHK 111
51: # define ID_ANGLE_GRP 112
52: # define ID_RADIAN_RBT 113
53: # define ID_DEGREE_RBT 114
54: # define ID_GRADIAN_RBT 115
55: // declare constants for the check box and radio button states
56: const BF_CHECKED = 1;
57: const BF_UNCHECKED = 0;
58: // declare angle-related constants
59: const double pi = 4* atan(1);
60: const double DegToRad = pi / 180;
61: const double GradToRad = 0.9 * DegToRad;
62:
63: // declare extended button class
64: class CxButton : public CButton
65: {
66: public:
67:
68:   BOOL Create(const char FAR* lpCaption, const RECT& rect,
69:               CWnd* pParentWnd, UINT nID)
70:   {
71:     return CButton::Create(lpCaption,
72:                            WS_CHILD ©¦WS_VISIBLE ©¦
73:                            WS_TABSTOP ©¦BS_PUSHBUTTON,
74:                            rect, pParentWnd, nID);
```

```
75:    }
76: };
77:
78: // declare check box class
79: class CCheckBox: public CButton
80: {
81: public:
82:    BOOL Create(const char FAR* lpCaption, const RECT& rect,
83:                CWnd* pParentWnd, UINT nID)
84:    {
85:       return CButton::Create(lpCaption,
86:                              WS_CHILD ©¦WS_VISIBLE ©¦
87:                              WS_TABSTOP ©¦BS_AUTOCHECKBOX,
88:                              rect, pParentWnd, nID);
89:    }
90:    // check control
91:    void Check() { SetCheck(BF_CHECKED); }
92:    // uncheck control
93:    void UnCheck() { SetCheck(BF_UNCHECKED); }
94: };
95:
96: // declare radio button class
97: class CRadioButton: public CButton
98: {
99: public:
100:    BOOL Create(const char FAR* lpCaption, const RECT& rect,
101:                CWnd* pParentWnd, UINT nID)
102:    {
103:       return CButton::Create(lpCaption,
104:                              WS_CHILD ©¦WS_VISIBLE ©¦
105:                              WS_TABSTOP ©¦
106:                              BS_AUTORADIOBUTTON,
107:                              rect, pParentWnd, nID);
108:    }
109:    // check control
110:    void Check() { SetCheck(BF_CHECKED); }
111:    // uncheck control
112:    void UnCheck()
113:       { SetCheck(BF_UNCHECKED); }
114: };
115:
116: // declare group box class
117: class CGroupBox: public CButton
118: {
119: public:
120:    BOOL Create(const char FAR* lpCaption, const RECT& rect,
121:                CWnd* pParentWnd, UINT nID)
122:    {
123:       return CButton::Create(lpCaption,
124:                              WS_CHILD ©¦WS_VISIBLE ©¦
125:                              WS_TABSTOP ©¦BS_GROUPBOX,
126:                              rect, pParentWnd, nID);
```

```
127:     }
128: };
129:
130:
131: class CxEdit:public CEdit
132: {
133: public:
134:
135:   BOOL GetLine(LPSTR ATextString, int nStrSize, int nLineNumber);
136:   int GetLineLength(int nLineNumber);
137: };
138:
139: class CWindowApp:  public CWinApp
140: {
141: public:
142:      virtual BOOL InitInstance();
143: };
144: // expand the functionality of CFrameWnd by deriving class
145: // CAppWindow
146: class CAppWindow:  public CFrameWnd
147: {
148: public:
149:
150:    CAppWindow();
151:    ~CAppWindow();
152:
153: protected:
154:    // declare the pointer to the various controls
155:    // first, the edit box controls
156:    CEdit*  Operand1Box;
157:    CEdit*  OperatorBox;
158:    CEdit*  Operand2Box;
159:    CEdit*  ResultBox;
160:    CEdit*  ErrMsgBox;
161:    CEdit*  VariableBox;
162:    // then the static text control
163:    CStatic*  Operand1Txt;
164:    CStatic*  OperatorTxt;
165:    CStatic*  Operand2Txt;
166:    CStatic*  ResultTxt;
167:    CStatic*  ErrMsgTxt;
168:    CStatic*  VariableTxt;
169:    // then the push buttons
170:    CxButton*  CalcBtn;
171:    CxButton*  StoreBtn;
172:    CxButton*  ExitBtn;
173:    // then the angle group box
174:    CGroupBox*  AngleModeGrp;
175:    // then the radio buttons
176:    CRadiobutton*  RadianRbt;
177:    CRadiobutton*  DegreeRbt;
178:    CRadiobutton*  GradianRbt;
```

```
179:    // then the check boxes
180:    CCheckBox* AutoVarSubstChk;
181:    CCheckBox* EchoGroupChk;
182:    // math error flag
183:    BOOL bInError;
184:    // the factor that converts between angles the
185:    // currently selected angle mode and radians
186:    double angleFactor;
187:
188:    // handle clearing the edit controls
189:    afx_msg void CMClearBox();
190:    // handle clearing the check box controls
191:    afx_msg void CMClearChk();
192:    // handle the calculation
193:    afx_msg void HandleCalcBtn();
194:    // handle the calculation
195:    afx_msg void CMCalc() {HandleCalcBtn(); }
196:    // handle storing the result in a variable
197:    afx_msg void HandleStoreBtn();
198:    // handle storing the result in a variable
199:    afx_msg void CMStore() {HandleStoreBtn(); }
200:    // handle exiting the application
201:    afx_msg void HandleExitBtn() {SendMessage(WM_CLOSE);}
202:    // handle exiting the application
203:    afx_msg void CMExit() {SendMessage(WM_CLOSE);}
204:    // handle exiting the application
205:    afx_msg void CMExit() {SendMessage(WM_CLOSE);}
206:    // handle the Angle Mode group box message
207:    // this member is called only by the radio button
208:    // controls; it therefore need not be included in the
209:    // message map
210:    void HandleAngleModeGrp();
211:    // Handle selecting the Radians radio button
212:    afx_msg void HandleRadianRbt();
213:    // Handle selecting the Degrees radio button
214:    afx_msg void HandleDegreeRbt();
215:    // Handle selecting the Gradians radio button
216:    afx_msg void HandleGradianRbt();
217:    // enale a push button control
218:    virtual void EnableButton(CxButton* pBtn)
219:      { pBtn->EnableWindow(TRUE);}
220:    // disable a push button control
221:    virtual void DisableButton(CxButton* pBtn)
222:      { pBtn->EnableWindow(FALSE);}
223:    // handle creating the control
224:    afx_msg int OnCreate(LPCREATESTRUCT lpCS);
225:    // handle closing the window
226:    afx_msg void OnClose();
227:    // initialize the instances of CAppWindow
228:    virtual void InitAppWindow();
229:    // return a reference to member r based on individual
230:    // coordinates and dimensions
```

```
231:    void makeRect(int nX, int nY, int nW, int nH, Crect& r);
232:    // obtain a number of a Variable edit box line
233:    double getVar(int nLineNum);
234:    // store a number in the selected text of
235:    // the Variable edit box line
236:    void putVar(double x);
237:    // declare the iterator for the check boxes
238:    void ForEachCheckBox(void (FAR * f)(CCheckBox* ));
239:    // declare the iterator for the edit controls
240:    void ForEachEditCtl(void (FAR * f)(CEdit* ));
241:
242:    // declare message map macro
243:    DECLARE_MESSAGE_MAP();
244: };
245:
246: // declare an iterated function to clear all edit control
247: // except the Variables edit control
248: void FAR ClearEditControls(CEdit*  P)
249: { P- > SetWindowText("");}
250:
251: // declare an iterated function to clear all edit boxes
252: void FAR ClearCheckBoxes(CCheckBox*  P)
253: { P- > UnCheck(); }
254:
255: BOOL CxEdit::GetLine(LPSTR ATextString, int nStrSize,
256:                      int nLineNumber)
257: {
258:    int nBytesCopied;
259:    BOOL bSuccess;
260:
261:    if (nStrSize < =  0)
262:       return FALSE;
263:    bSuccess =  (nStrSize > =  GetLineLength(nLineNumber) +  1) ? TRUE
264:                                                               : FALSE;
265:    if (nStrSize < =  1)
266:    {
267:       ATextString[0] =  '\0';
268:       return bSuccess;
269:    }
270:
271:    ((WORD FAR* )ATextString)[0] =  nStrSize;
272:
273:    nBytesCopied =  (WORD) (SendMessage(EM_SETLINE, nLineNumber,
274:                                        long(ATextString)));
275:    if (nBytesCopied)
276:    {
277:       // Windows return non-null-terminated string
278:       AtextString[nBytesCopied] =  '\0';
279:       return bSuccess;
280:    }
281:    else
282:       return FALSE;
```

```
283: }
284:
285: int CxEdit::GetLineLength(int nLineNumber)
286: {
287:    int nStartPos = -1;
288:
289:    if (nLineNumber > -1)
290:       nStartPos = LineIndex(nLineNumber);
291:    return (WORD) SendMessage(EM_LINELENGTH, nStartPos);
292: }
293:
294: CAppWindow::CAppWindow()
295: {
296:    // load accelerator resources
297:    LoadAccelTable("BUTTONS");
298:    // create the window
299:    Create (NULL,
300:        "Command-Oriented Calculator Application (COCA) Version 2",
301:         WS_OVERLAPPEDWINDOW ¦WS_MAXIMIZE,
302:           rectDefault, NULL, "EXITMENU");
303:    // perform initialization
304:    InitAppWindow();
305:    // clear the bInError flag
306:    bInError = FALSE;
307: }
308:
309: CAppWindow::~CAppWindow()
310: {
311:    // delete the controls
312:    delete Operand1box;
313:    delete OperatorBox;
314:    delete Operand2Box;
315:    delete ResultBox;
316:    delete ErrMsgBox;
317:    delete VariableBox;
318:    delete Operand1Txt;
319:    delete OperatorTxt;
320:    delete Operand2Txt;
321:    delete ResultTxt;
322:    delete ErrMsgTxt;
323:    delete VariableTxt;
324:    delete CalcBtn;
325:    delete StoreBtn;
326:    delete ExitBtn;
327:    delete AngleModeGrp;
328:    delete RadianRbt;
329:    delete DegreeRbt;
330:    delete GradianRbt;
331:    delete AutoVarSubstChk;
332:    delete EchoGroupChk;
333: }
334:
```

```
335: int CAppWindow::OnCreate(LPCREATESTRUCT lpCS)
336: {
337:    char s[81];
338:    char bigStr[6 * MAX_MEMREG + 1];
339:    char chC;
340:    int nX0 = 20;
341:    int nY0 = 30;
342:    int nx = nX0, nY = nY0;
343:    int nX1, nY1;
344:    CRect r;
345:    DWORD StatStyle = WS_CHILD ©¦WS_VISIBLE ©¦SS_LEFT;
346:    DWORD BoxStyle = WS_CHILD ©¦WS_VISIBLE ©¦WS_TABSTOP ©¦WS_BORDER
347:                     ©¦ES_LEFT ©¦ES_AUTOHSCROLL ©¦ES_UPPERCASE;
348:
349:    // create the first set of labels for the edit boxes
350:    makeRect(nX, nY, Wlbl, Hlbl, r);
351:    Operand1Txt = new CStatic();
352:    Operand1Txt->Create("Operand1", StatStyle, r, this, -1);
353:    nX += Wlbl + LblHorzSpacing;
354:    makeRect(nX, nY, Wlbl, Hlbl, r);
355:    OperatorTxt = new CStatic();
356:    OperatorTxt->Create("Operand1", StatStyle, r, this, -1);
357:    nX += Wlbl + LblHorzSpacing;
358:    makeRect(nX, nY, Wlbl, Hlbl, r);
359:    Operand2Txt = new CStatic();
360:    Operand2Txt->Create("Operand2", StatStyle, r, this, -1);
361:    nX += Wlbl + LblHorzSpacing;
362:    makeRect(nX, nY, Wlbl, Hlbl, r);
363:    ResultTxt = new CStatic();
364:    ResultTxt->Create("Result", StatStyle, r, this, -1);
365:
366:    // create the operand1, operator, operand2, and result
367:    // edit boxes
368:    nX = nX0;
369:    nY += Hlbl + LblVertSpacing;
370:    makeRect(nX, nY, Wbox, Hbox, r);
371:    Operand1Box = new CEdit();
372:    Operand1Box->Create("BoxStyle", r, this, ID_OPERAND1_EDIT);
373:    Operand1Box->LimitText(); // set no limit for text
374:    nX += Wbox + BoxHorzSpacing;
375:    makeRect(nX, nY, Wbox, Hbox, r);
376:    OperatorBox = new CEdit();
377:    OperatorBox->Create(BoxStyle, r, this, ID_OPERATOR_EDIT);
378:    OperatorBox->LimitText(); // set no limit for text
379:    nX += Wbox + BoxHorzSpacing;
380:    makeRect(nX, nY, Wbox, Hbox, r);
381:    Operand2Box = new CEdit();
382:    Operand2Box->Create(BoxStyle, r, this, ID_OPERAND2_EDIT);
383:    Operand2Box->LimitText(); // set no limit for text
384:    nX += Wbox + BoxHorzSpacing;
385:    makeRect(nX, nY, Wbox, Hbox, r);
386:    ResultBox = new CEdit();
```

```
387:    ResultBox- > Create(BoxStyle, r, this, ID- RESULT- EDIT);
388:    ResultBox- > LimitText(); // set no limit for text
389:    // create the static text and edit box for the error message
390:    nX = nX0;
391:    nY + = Hbox + BoxVertSpacing;
392:    makeRect(nX, nY, Wlbl, Hlbl, r);
393:    ErrMsgTxt = new CStatic();
394:    ErrMsgTxt- > Create("Error Message", StatStyle, r, this, - 1);
395:    nX + = Hlbl + LblVertSpacing;
396:    makeRect(nX, nY, WLongbox, Hbox, r);
397:    ErrMsgBox = new CEdit();
398:    ErrMsgBox- > Create(BoxStyle, r, this, ID- ERRMSG- EDIT);
399:    ErrMsgBox- > LimitText(); // set no limit for text
400:    // create the static text and edit box for the single-letter
401:    // variable selection
402:    nX + = Hbox + BoxVertSpacing;
403:    makeRect(nX, nY, Wlbl, Hlbl, r);
404:    VariableTxt = new CStatic();
405:    VariableTxt- > Create("Variables", StatStyle, r, this, -1);
406:    nX + = Hlbl + LblVertSpacing;
407:    bigstr[0] = '\0';
408:    // build the initial contents of the Variable edit box
409:    for (chC = 'A'; chC < = 'Z'; chC+ + ) {
410:      sprintf(s, "%c: 0\r\n", chC);
411:      strcat(bigStr, s);
412:    }
413:    makeRect(nX, nY, Wvarbox, Hvarbox, r);
414:    VariableBox = new CEdit();
415:    VariableBox- > Create(BoxStyle ©¦ES- MULTILINE ©¦WS- HSCROLL
416:                    ©¦WS- VSCROLL ©¦ES- AUTOVSCROLL,
417:                        r, this, ID- VARIABLE- EDIT);
418:    VariableBox- > LimitText(); // set no limit for text
419:    VariableBox- > SetWindowText(bigStr);
420:
421:    // create the Calc push button
422:    nX + = Wvarbox + BtnHorzSpacing;
423:    nX1 = nX;
424:    nY1 = nY;
425:    makeRect(nX, nY, Wbtn, Hbtn, r);
426:    CalcBtn = new CxButton();
427:    CalcBtn- > Create("&Calc", r, this, ID- CALC- BTN);
428:
429:    // create the Store Btn
430:    nX + = Wbtn + BtnHorzSpacing;
431:    makeRect(nX, nY, Wbtn, Hbtn, r);
432:    StoreBtn = new CxButton();
433:    StoreBtn- > Create("&Store", r, this, ID- STORE- BTN);
434:
435:    // create the Exit Btn
436:    nX + = Wbtn + BtnHorzSpacing;
437:    makeRect(nX, nY, Wbtn, Hbtn, r);
438:    ExitBtn = new CxButton();
```

```
439:    ExitBtn- > Create("&Exit", r, this, ID- EXIT- BTN);
440:
441:    // create the "Substitute variable name" check box
442:    nX = nX0;
443:    nY + = Hvarbox + BoxVertSpacing;
444:    makeRect(nX, nY, Wchk, Hchk, r);
445:    AutoVarSubstChk = new CCheckBox();
446:    AutoVarSubstChk- > Create("Substitute variable name", r,
447:                                this, ID- SUBST- CHK);
448:
449:    // create the "Echo Group box message" check box
450:    nX + = Hchk + ChkVertSpacing;
451:    makeRect(nX, nY, Wchk, Hchk, r);
452:    EchoGroupChk = new CCheckBox();
453:    EchoGroupChk- > Create("Echo Group box message", r,
454:                            this, ID- ECHO- CHK);
455:
456:    // create the Angle Mode group box
457:    nY + = nY1 + Hbtn + BtnVertSpacing;
458:    nX = nX1;
459:    makeRect(nX, nY, Wgrp, Hgrp, r);
460:    AngleModeGrp = new CGroupBox();
461:    AngleModeGrp- > Create("Angle Mode", r, this, ID- ANGLE- GRP);
462:
463:    // create the Radians radio button
464:    nY + = RbtVertSpacing;
465:    makeRect(RbtLeftMargin + nX, nY, Wrbt, Hrbt, r);
466:    RadianRbt = new CRadioButton();
467:    RadianRbt- > Create("Radians", r, this, ID- RADIAN- RBT);
468:
469:    // create the Degress radio button
470:    nY + = RbtVertSpacing;
471:    makeRect(RbtLeftMargin + nX, nY, Wrbt, Hrbt, r);
472:    DegreeRbt = new CRadioButton();
473:    DegreeRbt- > Create("Degrees", r, this, ID- DEGREE- RBT);
474:
475:    // create the Gradians radio button
476:    nY + = RbtVertSpacing;
477:    makeRect(RbtLeftMargin + nX, nY, Wrbt, Hrbt, r);
478:    GradianRbt = new CRadioButton();
479:    GradianRbt- > Create("Gradians", r, this, ID- GRADIAN- RBT);
480:
481:    return CFrameWnd::OnCreate(lpCS);
482: }
483:
484: void CAppWindow::InitAppWindow()
485: {
486:
487:    // disable the Store button control
488:    EnableButton(StoreBtn);
489:
490:    // check the Degrees radio button
```

```
491:    DegreeRbt->Check();
492:    angleFactor = DegToRad;
493:
494:    // check the "Echo Group message box" check button
495:    EchoGroupChk->Check();
496: }
497:
498: void CAppWindow::CMClearBox()
499: {
500:    ForEachEditCtl(ClearEditControls);
501: }
502:
503: void CAppWindow::CMClearChk()
504: {
505:    ForEachEditBox(ClearCheckBoxes);
506: }
507:
508: void CAppWindow::HandleCalcBtn()
509: {
510:    double x, y, z, result;
511:    char opStr[MaxEditLen+1];
512:    char s[MaxEditLen+1];
513:
514:    // convert the string in the Result box to a double
515:    ResultBox->GetWindowText(s, MaxEditLen);
516:    result = atof(s);
517:
518:    // obtain the string in the Operand1 edit box
519:    Operand1Box->GetWindowText(s, MaxEditLen);
520:    // does the Operand1Box contain the name
521:    // of a single-letter variable?
522:    if (isalpha(s[0])) {
523:       // obtain value from the Variable edit control
524:       x = getVar(s[0] - 'A');
525:       // substitute the variable name with its value
526:       if (AutoVarSubstChk->GetCheck() == BF_CHECKED) {
527:          sprintf(s, "%g", x);
528:          Operand1Box->SetWindowText(s);
529:       }
530:    }
531:    // translate the # character into the value in the Result box
532:    else if (s[0] == '#')
533:       x = result;
534:    else
535:       // convert the string in the edit box
536:       x = atof(s);
537:
538:    // obtain the string in the Operand2 edit box
539:    Operand2Box->GetWindowText(s, MaxEditLen);
540:    // does the Operand2Box contain the name
541:    // of a single-letter variable?
542:    if (isalpha(s[0])) {
```

```
543:      // obtain value from the Variable edit control
544:      y = getVar(s[0] - 'A');
545:      // substitute the variable name with its value
546:      if (AutoVarSubstChk->GetCheck() == BF_CHECKED) {
547:         sprintf(s, "%g", y);
548:         Operand2Box->SetWindowText(s);
549:      }
550:   }
551:   // translate the # character into the value in the Result box
552:   else if (s[0] == '#')
553:      y = result;
554:   else
555:      // convert the string in the edit box
556:      y = atof(s);
557:
558:   // obtain the string in the Operator edit box
559:   OperatorBox->GetWindowText(opStr, MaxEditLen);
560:
561:   // clear the error message box
562:   ErrMsgBox->SetWindowText("");
563:   bInError = FALSE;
564:
565:   // determine the requested operation
566:   if (strlen(opStr) == 1) {
567:   if (strcmp(opStr, "+") == 0)
568:      z = x + y;
569:   else if (strcmp(opStr, "-") == 0)
570:      z = x - y;
571:   else if (strcmp(opStr, "*") == 0)
572:      z = x * y;
573:   else if (strcmp(opStr, "/") == 0)
574:      if (y != 0)
575:         z = x / y;
576:      else {
577:         z = 0;
578:         bInError = TRUE;
579:         ErrMsgBox->SetWindowText("Division-by-zero error");
580:      }
581:   }
582:   else if (strcmp(opStr, "^") == 0) {
583:      if (x > 0)
584:         z = exp(y * log(x));
585:      else {
586:         bInError = TRUE;
587:         ErrMsgBox->SetWindowText(
588:            "Cannot raise the power of a negative number");
589:      }
590:   }
591:   else {
592:         bInError = TRUE;
593:         ErrMsgBox->SetWindowText("Invalid operator");
594:   }
```

```
595:   }
596:   else if (strcmp(opStr, "SIN") == 0) {
597:    z = sin(angleFactor * x);
598:   }
599:   else if (strcmp(opStr, "COS") == 0) {
600:    z = cos(angleFactor * x);
601:   }
602:   else if (strcmp(opStr, "TAN") == 0) {
603:    z = tan(angleFactor * x);
604:   }
605:   else if (strcmp(opStr, "ASIN") == 0) {
606:    if (fabs(x) <= 1)
607:       z = asin(x) / angleFactor;
608:    else {
609:       bInError = TRUE;
610:          ErrMsgBox->SetWindowText(
611:             "Invalid argument of the asin(x) function");
612:    }
613:   }
614:   else if (strcmp(opStr, "ACOS") == 0) {
615:    if (fabs(x) <= 1)
616:       z = cos(x) / angleFactor;
617:    else {
618:       bInError = TRUE;
619:          ErrMsgBox->SetWindowText(
620:             "Invalid argument of the acos(x) function");
621:    }
622:   }
623:   else if (strcmp(opStr, "ATAN") == 0) {
624:    z = atan(x) / angleFactor;
625:   }
626:    else {
627:       bInError = TRUE;
628:          ErrMsgBox->SetWindowText("Invalid math function");
629:   }
630:
631:   // display the result if no error has occurred
632:   if (! bInError) {
633:      sprintf(s, "%g", z);
634:      ResultBox->SetWindowText(s);
635:      // enable the Store button
636:      EnableButton(StoreBtn);
637:   }
638:   else
639:      // disable the Store button
640      DisableButton(StoreBtn);
641:   }
642:
643: void CAppWindow::HandleStoreBtn()
644: {
645:      char result[MaxEditLen+1];
646:
```

```
647:     // get the string in the Result edit box
648:     ResultBox->GetWindowText(result, MaxEditLen);
649:
650:     // store the result in the selected text of
651:     // the Variable edit box
652:     putVar(atof(result));
653: }
654:
655: void CAppWindow::HandleAngleModeGrp()
656: {
657:     char angleStr[81];
658:
659:     // exit if the EchoGroup check box is not checked
660:     if (EchoGroupChk->GetCheck() != BF_CHECKED) return;
661:     // build the text of the message
662:     if (DegreeRbt->GetCheck() == BF_CHECKED)
663:       sprintf(angleStr, "1 radian = %g degrees", 1 / DegToRad);
664:     else if (GradianRbt->GetCheck() == BF_CHECKED)
665:       sprintf(angleStr, "1 radian = %g gradians", 1 / GradToRad);
666:     else
667:       sprintf(angleStr, "1 radian = %g degrees = %g gradians",
668:               1 / DegToRad, 1 / GradToRad);
669:     MessageBox(angleStr, "Group Box Message",
670:               MB_OK ©¦MB_ICONINFORMATION);
671: }
672:
673: void CAppWindow::HandleRadianRbt()
674: {
675:   angleFactor = 1;
676:   HandleAngleModeGrp();
677: }
678:
679: void CAppWindow::HandleGradianRbt()
680: {
681:   angleFactor = DegToRad;
682:   HandleAngleModeGrp();
683: }
684:
685: void CAppWindow::HandleGradianRbt()
686: {
687:   angleFactor = GradToRad;
688:   HandleAngleModeGrp();
689: }
690:
691: void CAppwindow::makeRect(int X, int Y, int W, int H, CRect& r)
692: {
693:   r.top = Y;
694:   r.left = X;
695:   r.bottom = Y + H;
696:   r.right = X + W;
697: }
698:
```

```
699: double CAppWindow::getVar(int nLineNum)
700: {
701:    int nLineSize;
702:    char s[MaxEditLen+ 1];
703:
704:    if (nLineNum > = MAX- MEMERG) return 0;
705:    // get the size of the target line
706:    nLineSize = VariableBox- > GetLineLength(nLineNum);
707:    // get the line
708:    VariableBox- > GetLine(s, nLineSize, nLineNum);
709:    // delete the first three characters
710:    strcpy(s, (s+ 3));
711:    // return the number stored in the target line
712:    return atof(s);
713: }
714:
715: void CAppWindow::putVar(double x)
716: {
717:    DWORD dwSelPos;
718:    WORD wStartPos, wEndPos;
719:    int nLineNum;
720:    int nLineSize;
721:    char s[MaxEditLen+ 1];
722:
723:    // locate the character position of the cursor
724:    dwSelPos = VariableBox- > GetSel();
725:    wStartPos = LOWORD(dwSelPos);
726:    wEndPos = HIWORD(dwSelPos);
727:    // turn off the selected text
728:    if (wStartPos ! = wEndPos) {
729:       dwSelPos = MAKELONG(wStartPos, wStartPos);
730:       VariableBox- > SetSel(dwSelPos);
731:    }
732:    // get the line number where the cursor is located
733:    nLineNum = VariableBox- > LineFromChar(- );
734:    // get the line size of line nLineNum
735:    nLineSize = VariableBox- > GetLineLength(nLineNum);
736:    // obtain the text of line nLineNum
737:    VariableBox- > GetLine(s, nLineSize, nLineNum);
738:    // build the new text line
739:    sprintf(s, "%c: %g", s[0], x);
740:    // get the character positions for the deleted line
741:    wStartPos = (WORD) (VariableBox- > LineIndex(- 1));
742:    wEndPos = (WORD) (wStartPos + VariableBox- > LineLength(- 1));
743:    // select the current line
744:    dwSelPos = MAKELONG(wStartPos, wEndPos);
745:    VariableBox- > SetSel(dwSelPos);
746:    // replace the current line with the new line
747:    VariableBox- > ReplaceSel(s);
748: }
749:
750: void CAppWindow::OnClose()
```

```
751: {
752:    if (MessageBox("Want to close this application",
753:                   "Query", MB_YESNO ©¦MB_ICONQUESTION) == IDYES)
754:       DestroyWindow();
755: }
756:
757: void CAppWindow::ForEachCheckBox(void(FAR * f)(CCheckBox* ))
758: {
759:    CCheckBox*  P;
760:
761:    for (int nID = ID_SUBST_CHK; nID <= ID_ECHO_CHK; nID++ ) {
762:       // get the pointer to the next button
763:       P = (CCheckBox* ) (this->GetDlgItem(nID));
764:       // process the button
765:       (* f)(p);
766:    }
767: }
768:
769: void CAppWindow::ForEachEditCtl(void (FAR * f)(CEdit* ))
770: {
771:    CEdit*  P;
772:
773:    for (int nID = ID_OPERAND1_EDIT; nID <= ID_ERRMSG_EDIT; nID++ ) {
774:       // get the pointer to the next button
775:       P = (CEdit* ) (this->GetDlgItem(nID));
776:       // process the button
777:       (* f)(p);
778:    }
779: }
780:
781: BEGIN_MESSAGE_MAP(CAppWindow, CFrameWnd)
782:    ON_COMMAND(CM_CLEARBOX, CMClearBox)
783:    ON_COMMAND(CM_CLEARCHK, CMClearChk)
784:    ON_BN_CLICKED(ID_CALC_BTN, HandleCalcBtn)
785:    ON_COMMAND(ID_CALC_BTN,CMCalc)
786:    ON_BN_CLICKED(ID_STORE_BTN, HandleStoreBtn)
787:    ON_COMMAND(ID_STORE_BTN, CMStore)
788:    ON_BN_CLICKED(ID_EXIT_BTN, HandleExitBtn)
789:    ON_COMMAND(ID_EXIT_BTN, CMExit)
790:    ON_COMMAND(CM_EXIT, OnExit)
791:    ON_BN_CLICKED(ID_RADIAN_RBT, HandleRadianRbt)
792:    ON_BN_CLICKED(ID_DEGREE_RBT, HandleDegreeRbt)
793:    ON_BN_CLICKED(ID_GRADIAN_RBT, HandleGradianRbt)
794:    ON_WM_CREATE()
795:    ON_WM_KEYDOWN()
796:    ON_WM_CLOSE()
797: END_MESSAGE_MAP()
798:
799: // Construct the CWindowApp's m_pMainWnd data member
800: BOOL CWindowApp::InitInstance()
801: {
802:    m_pMainWnd = new CAppWindow();
```

```
803:    m_pMainWnd->ShowWindow(m_nCmdShow);
804:    m_pMainWnd->UpdateWindow();
805:    return TRUE;
806: }
807:
808: // application's constructor initializes and runs the app
809: CWindowApp WindowApp;
```

图 17.1 中显示了程序 CTLGRP1.EXE 的一个运行范例。

图 17.1　程序 CTLGRP1.EXE 的一个运行范例

分　析

应用程序中声明了如下的控制组:

- 第 10 到 37 行的一组常量用以指定各种控制的大小和它们之间的间隔。
- 第 43 到 54 行中声明的为基于宏的常量,用以定义了各种控制的 ID 值。
- 第 56 到 57 行的常量,用于选中框和圆按钮的状态。
- 第 59 到 61 行的常量,指定了 pi 值、弧度与度以及弧度与百分度的角度转换因子。

在程序 CTLGRP1.CPP 中定义了如下的类:

- CxButton 类,它是 CButton 的子孙类,用于建立下压按钮控制(其声明从第 64 行开始)模型。
- CCheckBox 类,它是 CButton 的子孙类,用于建立选中框控制(其声明从第 79 行开始)模型。
- CRadioButton,它是 CButton 的派生类,用于建立圆按钮控制(其声明从第 97 行开始)模型。
- CGroupBox 类,它是 CButton 的派生类,用于建立分组框控制(其声明从第 117 行

开始) 模型。

- CxEdit 类, 它是 CEdit 的派生类, 用于建立增强版本的编辑控制(其声明从第 131 行开始) 模型。
- CWindowApp 应用程序类(其声明从第 139 行开始) 。
- CAppWindow 窗口类(其声明从第 146 行开始) 。

类 CCheckBox 和 CRadioButton 声明了成员函数 Check 和 UnCheck, 用于清除和设定控制的选中状态。

窗口类 CAppWindow 第 156 到 186 行声明了很多数据成员, 它们大多为指向应用程序中使用的各种控制实例的指针。应用程序加入数据成员 angleFactor 用于保存当前选中的角度制与弧度之间的转换因子。

CAppWindow 类定义了一个构造函数、一个析构函数和若干成员函数, 以支持完整的程序操作和特殊初始化。成员函数 OnCreate 协助构造函数加载菜单资源和建立各种控制。函数 OnCreate 中的语句很重要, 它们负责建立选中框、分组框和圆按钮。每个 CCheckBox 的实例有一个唯一的控制 ID、标题和坐标。这些 CCheckBox 实例建立时不带 WS_GROUP 风格, 表明每个选中框都是在分组框之外建立的。分组框实例具有唯一的 ID、标题和坐标。每个圆按钮也具有唯一的 ID、标题和坐标。注意这些圆按钮的父窗口是主窗口而不是分组框, 虽然看上去它们位于分组框内。

CAppWindow 类声明了各种成员函数用于实现程序的运算, 函数 CMClearBox 清除所有的编辑控制的文本, 但 Variables 编辑控制除外。函数利用成员函数作为迭代算子, 执行请求的操作。类似地, 成员函数 CmClearChk 把成员函数 forEachCheckBox 作为迭代算子, 清除选中框。

成员函数 HandleCalcBtn 执行计算和字符替换。本成员函数完成以下任务:

- 将 Result 编辑控制的字符串转换成一个 double 型数, 并把它保存在局部变量中。
- 获取变量 X 中所存的的第一操作数的值。Operand1 编辑控制中的字符串或由数值和单字名变量名组成, 或为# 字符:

 - 支持第一个选项的语句判定 Operand1 框中是否包含一个字母。如果条件成立, 调用成员函数 getVar, 把结果赋给变量 X。另外, 代码还检测 Substitute(替换) 变量名选中框是否选中, 如选中, Operand1 框的变量名由它的值取代。
 - 支持第二个选项的语句用# 字符与 Operand1 框中的第一个字符比较。如两者相同, 将存储在局域变量结果中的值赋给变量 X。
 - 支持第三个选项的语句把 Operand1 编辑框中的字符串转换为一个 double 型数, 并把它赋给变量 X。

- 获取第二操作数的方式与第一个类似(第二操作数在局部变量 y 中) 。
- 从 Operator 编辑框中获得运算符或函数。
- 执行要求的运算或函数计算, 其中用到一系列的 if 语句来判定要求的操作。语句中还包括错误检则, 如果有错误发生, 函数给数据成员 bInError 赋值为 TRUE, 并在出错消息编辑控制中显示出错消息。

· 如果没有错误发生,在 Result 编辑框中显示结果。

成员函数 CMCalcBtn, HandleStoreBtn, CMStoreBtn, HandleExitBtn, CMExitBtn, putVar 和 gerVar 与程序文件 CTLBTN1.CPP 中的 COCA 第二版一样。

第 655 到 671 行定义了成员函数 HandleAngleModeGrp,它由圆按钮的消息响应函数调用。函数先校验 Echo Group 消息选中框是否未被选中。如果未选中,成员函数退出;否则,函数判定哪个圆按钮选中,据此产生一条角度转换消息,函数调用 MessageBox 函数显示这条消息。

第 673 行到 689 行定义了成员函数 HandleRadianRbt, HandleDegreeRbt 和 HandleGradianRbt,它们分别响应三个圆按钮发出的通知消息,每个成员函数都把一个角度转换因子赋给数据成员 angleFactor,然后调用函数 HandleAngleModeGrp,调用这些成员函数检查圆按钮的选中状态比用成员函数 HandleCalcBtn 更有效。使用这种方法,只有在选择一个新的角度制时,才能改变 angleFactor 的值。

> **注　意**　运行本应用程序并选中另一个圆按钮时,先前的圆按钮自动地取消选中状态。看来 MFC 库把一个窗口所属的圆按钮看成一个简单的组,这样,在程序中就不需要去代码来去掉先前圆按钮的选中标记。

CAppWindow 类包括了 InitAppWindow 成员函数,它初始化以下控制:

· 使 Store 按钮失效。
· 选中 Degrees 圆按钮。
· 选中 Echo Group 消息框的选中框。

> **注　意**　窗口类在第 757 到 779 行定义了两个迭代函数,名为 ForEachEditCtl 和 ForEachCheckBox,两个函数都用 for 循环在整个目标控制 ID 范围内重复操作。每一个循环迭代都根据 ID 值而获得控制的指针,然后将迭代函数提供给控制。迭代(非成员)函数是 ClearEditControl 和 ClearCheckBoxes,在第 246 到 253 行定义。每个函数都有一条执行寻找任务的语句。本例演示了能提供更多控制且代码更少的专门的迭代算子。这是与另一个主要供应商的 C+ + 编译器比较而得出的。

小　　结

今天这一课讨论了特殊的开关控制、分组框、选中框和圆按钮,概括如下:

· 选中框和圆按钮控制用作软件开关。
· 对选中框和圆按钮控制,可以设定和查询它们的选中状态。
· 可以响应这些控制发送给它们父窗口的通知消息。
· 分组框把圆按钮和选中框组织起来。

· 开关控制可有选择地操作。

第 18 天将介绍滚动条、列表框和组合框控制，这些控制是有用的选择器，因为它们可以允许从值的列表或范围中进行选择。

问　与　答

问：选中框有状态 BF_UNCHECKED，BF_CHECKED 和 BF_GRAYED。第三种状态表示什么？

答：可以用 BF_GRAYED 状态表示“不用管”或不确定状态。

问：可以利用出现在程序 CTLGRP1.CPP 中的类 CxButton，CRadioButton 和 CheckBox 吗？为什么这些类不是 MFC 库的一部分呢？

答：使用这些类可以增加程序的可读性。因为能读出他们将产生的控制的类型。这些类不是 MFC 库中的一部分，这是由设计决定的，MFC 的开发者想把多种控制都挤进类 CButton 中。

问：如果把选中框控制放进分组控制中，会有什么不同？

答：把选中框控制放入一个分组控制中，会影响到控制的逻辑分组，对用户来说这将增强界面的吸引力。

作　　业

作业部分所提供的测验题有助于巩固理解所学内容，通过练习可以积累运用所学知识的经验。要尽量弄懂测验和练习的答案后再继续下一天的课程。附录 B 的“答案”中提供了解答。

测　验

1. 对还是错？
 一个选中框可替换一个分组框中的任意两个圆按钮。
2. 对还是错？
 当有 3 个或更多选项时，应使用分组控制中的圆按钮。
3. 对还是错？
 一组选中框与一个字节或字中的位相似。
4. 对还是错？
 同一分组控制中的圆按钮相互排斥。

练　习

扩充程序 CTLGRP1.CPP，加入更多的数学函数，如双曲线函数 sinh，cosh 和 tanh 和它们的反函数。

第18天

列 表 框

列表控制是一种很方便的输入工具,它提供一些项供用户选择。由于列表控制能把我们从死记硬背列表成员中解救出来,因而得以普遍使用。特别是计算机程序需要精确地拼写时更是显示了列表的益处。使用 DOS 程序的经历表明,各种 DOS 工具中,用列表显示文件和路径的,比起那些做法与之相反的,即假设你知道所有文件的文件名和路径的工具,更容易使用,也更容易博得大家的好感。使用列表控制已逐渐成为保存信息的基本方法。今天这一课,我们讨论单项选择和多项选择列表框,学习以下内容:

- 列表框控制
- 处理单项选择列表框
- 处理多重选择列表框

18.1 列表框控制

列表框通常是带有竖直滚动条的一个框架,当点中选择的项时,选中项就变亮。Microsoft 公司倡导在建立选项时遵循如下原则:

- 用鼠标单击来选择一个新的或一个附加选项时,需用一个单独的按钮来激活所选项。
- 而用鼠标双击,则不但可选择,而且可激活。

新 词

列表框(list box)
列表框是一个输入控制,它允许应用程序使用者从项的列表中选择。

一个列表框控制只有在建立该控制并指定了多项选择风格时,它才支持多项选择。在此方式下进行多项选择是很方便的。例如,选择多个要删除的文件,便能加速处理过程并减少你的干预。

CListBox 类

Microsoft 基本类(MFC)库提供了 CListBox 类,它是 CWnd 的派生类,以便利用列表控制。CListBox 类有一系列成员函数,可简便地操作和查询列表框的内部和已选项。和 MFC 库的其他许多类一样,类 CListBox 使用缺省构造函数和成员函数 Create 建立列表框实例。

语　法

函数 Create

函数 Create 定义如下:

```
BOOL Create (DWORD dwStyle, // control style
const RECT& rect, // location and dimensions
CWnd* pParentWnd, // pointer to parent window
UINT nID);       // control ID
```

参数 dwStyle 确定列表框控制的风格,列表框风格包括一般的 WS_xxxx 窗口风格和特殊的 LBX_xxxx 列表框风格(在表 18.1 中)。参数 rect 指定控制的大小和位置,参数 nParentWnd 为指向父窗口的指针。参数 nID 给滚动条实例指定唯一的 ID。

示　例:

```
CListBox * pList;
pList-> Create(LBS_STANDARD, rectDefault, this, ID_FILES_LST);
```

注　意　LBS_STANDARD 风格与 WS_BORDER, WS_VSCROLL, LBS_SORT 和 LBS_NOTIFY 风格等价。当删除风格 LBS_SORT 时,列表项不会自动地排序,这种列表允许按年代顺序维护表项。也可用降序排列项,当然,得自己负责按这种顺序维护列表项。若去掉 WS_VSCROLL 风格,则列表框不带竖直滚动条。下一节我们将提供一个使用这种列表框类型的演示程序,该程序实现多项选择列表框的同步滚动。

表 18.1　列表框控制风格

风　格	含　义
LBS_EXTENDEDSEL	允许使用 Shift 键扩展列表框的多重选择选项。
LBS_MULTICOLUMN	指定可水平滚动的多列列表框,需要使用 Windows API 函数(列数通过发送消息 LBN_SETCOLUMNWINDTH 设置)。
LBS_MULTIPLESEL	支持列表框的多重选择。
LBS_NOINTEGRALHEIGHT	禁止部分显示表项。
LBS_NOREDRAW	当选择改变时,禁止列表框更新(消息 WM_SETREDRAW 将更改此风格)。
LBS_NOTIFY	当单击或双击列表框时,通知父窗口。
LBS_SORT	指定插入列表框的项将自动以升序排列。
LBS_STANDARD	设定 WS_BORDER, WS_VSCROLL, LBS_SORT 和 LBS_NOTIFY 风格。
LBS_WANTKEYBOARDINPUT	当列表框获得输入焦点(应用程序可以操作列表框。中的项)且有键按下时,允许列表框属主处理接收 WM_VKEYTOITEM 或 WM_CHARTOITEM 消息。

CListBox 类允许用索引引用列表框中的项,第一项的索引为 0。CListBox 类提供了以下的成员函数来设置和查询普通的和选中的列表项:

- 成员函数 AddString 在列表框中增加一字符串。
- 成员函数 DeleteString 删除指定位置的列表项。
- 无参数成员函数 ResetContent 快速清除列表框控制里的列表串，此函数用于在建立新表之前重置一个列表框的内容。
- 成员函数 FindString 区分大小地搜寻列表框的一项。
- 无参数成员函数 GetCount 返回列表框中的项数，若有错返回 LB_ERR 值。
- 无参数成员函数 GetCurSel 返回单项选择列表框中的选中项位置，如没有选中项，函数返回负值。此函数只能用于单项选择列表框。
- 成员函数 GetSel 返回指定索引的列表项的选中状态。
- 无参数成员函数 GetSelCount 返回列表框中选中项的数目，对单项选择列表框，函数返回 LB_ERR。
- 成员函数 GetSelItem 返回多项选择列表框已选项的数目和位置。
- 重载成员函数 GetText 获得列表框中指定索引的表项。
- 成员函数 GetTextLen 返回列表中指定位置的表项的长度。
- 无参数成员函数 GetTopIndex 返回列表框中第一可见项的索引。
- 成员函数 InsertString 在列表框中插入一字符串。
- 成员函数 SelectString 选中与查找字符串匹配的列表项。
- 成员函数 SetCurSel 选一个新表项作为单项选择列表框的选中表项。
- 成员函数 SetItemRange 是允许在一次调用中选一个表项范围的另一个函数。
- 成员函数 SetSel 在一个多项选择列表框中建立或清除一个选中表项。
- 成员函数 SetTopIndex 设置列表框控制中第一个可见表项。
- 成员函数 Dir 是一个特殊的成员函数，它允许把文件名插入列表框中。

语　法

函数 AddString

成员函数 AddString 声明如下：

```
int AddString(const char FAR* lpItem)
```

参数 lpItem 指向要加入的字符串，函数返回控制中加入字符串的位置。如加入字符串时有错误发生，函数返回 LB_ERR 或 LB_ERRSPACE(内存耗尽错)。如设置了 LBS_SORT 风格，字串按列表顺序插入。如未设置 LBS_SORT 风格，所增加的字符串插入到表尾。

示　例：

```
Clistbox* pList;
// create control...
pList-> AddString("MS-DOS");
```

语　法

函数 DeleteString

成员函数 DeleteString 声明如下：

```
int DeleteString(int nIndex);
```

参数 nIndex 指定要删除的项的位置，函数返回剩下的表成员数目，如有错误发生，DeleteString 返回 LB_ERR。

示　例:

```
ClistBox *  pList;
// create control . . .
// add names to the list . . .
pList- > DeleteString("MS-DOS");
```

语　法

函数 FindString

成员函数 FindString **声明如下:**

```
int FindString(int nStartAfter, const char FAR*  lpItem) const;
```

参数 nStartAfter **为搜寻的列表框第一个成员的索引,参数** lpItem **指向要查找的字符串。这个函数搜寻整个表。从** nStartAfter **位置开始,如果需要的话可以从表头继续搜寻。当一个表成员与查找字符串匹配或搜索完整个表时则停止。如果** nStartAfter **为-** 1,**强制函数从表头开始搜索。函数返回匹配表项的位置,如未找到或发生错误,则返回** LB - ERR。

示　例:

```
ClistBox*  pList;
int nIdx;
// create control. . .
// add names to the list. . .
nIdx =  pList- > FindString(- 1, "MS-DOS");
```

注　意　FindString **所用的搜索方法很有趣,如指定的位置在最可能发生匹配的位置的前面,则会加快搜索速度,它的妙处在于,若指定的位置在要找的字符串的后面,也不会错过发现这个字符串的机会,因为函数将从表头继续搜索。**FindString **另一个好处在于能发现重复字符串。**

语　法

函数 GetSel

函数 GetSel **定义如下:**

```
int GetSel(int nIndex) const;
```

参数 nIndex **为查询的列表项的索引,如果表项被选中,则函数返回正值;如未选中则返回** 0;**如果有错误发生,则返回** LB - ERR。

示　例:

```
ClistBox *  pList;
// create control. . .
// add names to the list. . .
it(pList- > GetSel(0))
// process information
```

语　法

函数 GetSelItems

函数 GetSelItems **声明如下:**

```
int GetSelItems(int nMaxItems, LPINT rgIndex) const;
```

参数 nMaxItems 为指针 rgIndex 指向的数组的大小,参数 rgIndex 指向存有已选中项位置的整型数组。函数返回当前选中项的数目,对于单选列表框则返回 LB_ERR。

示　例:

```
const int MAX_ARRAY = 100;
CListBox* pList;
int intArr[MAX_ARRAY];
int nCount;
// create control ...
// add names to the list ...
nCount = pList->GetSelItems(MAX_ARRAY, intArr);
```

语　法

函数 GetText

函数 GetText 声明如下:

```
int GetText(int nIndex, char FAR* lpBuffer) const;
void GetText(int nIndex, CString& rString) const;
```

参数 nIndex 为要检索的项的索引,列表框第一项的索引为 0。参数 lpBuffer 指向接收检索项的缓存区。当然应该保证缓冲区有足够的空间用于保存检索到的项。参数 rString 为 CString 对象的引用,用于接收列表框项的一个拷贝。函数的第一种形式返回从列表框检索到的字符数。

示　例:

```
const STR_SIZE = 80;
char s[STR_SIZE + 1];
CListBox * pList;
// create control ...
// add names to the list ...
   if (pList->GetText(0,s) > 0)
   MessageBox(s, MB_OK, "Information");
```

语　法

函数 GetTextLen

函数 GetTextLen 声明如下:

```
int GetTextLen(int nIndex) const;
```

参数 nIndex 为目标列表项的索引。函数返回目标项的长度,如出错则返回 LB_ERR。

```
int nLen;
CListBox* pList;
// create control ...
// add names to the list ...
nLen = pList->GetTextLen(0);
```

语　法

函数 InsertString

函数 InsertString 声明如下:

```
int InsertString(int nIndex, const char FAR* lpItem);
```

参数 nIndex 为要求插入的位置,参数 lpItem 指向要插入的字符串。本函数返回实际插入位置,如出错则返回 LB_ERR。如果 nIndex 为-1,字符串添加到表尾。

示　例:

```
CListBox * pList;
// create control ...
// add names to the list ...
if (pList- > InsertString(0, "Windows") = = 0)
MessageBox("Inserted at index 0", MB- OK, "Information");
```

> **警　告**　通常,在设置了 LBS - SORT 风格的列表框中,不要用成员函数 InsertString。对排好序的列表框使用此函数时,很可能破坏列表的有序性。

语　法

函数 SelectString

函数 SelectString 声明如下:

```
int SelectString(int nStartAfter, const char FAR* lpItem);
```

SelectString 的参数和搜索机制与 FindString 一样,不同的是,SelectString 将选中与参数 lpItem 指向的字符串匹配的列表项。

示　例:

```
CListBox * pList;
int nIdx;
// create control ...
// add names to the list ...
nIdx = pList- > SelectString(- 1, "MS-DOS");
```

语　法

函数 SelItemRange

函数 SelItemRange 声明如下:

```
int SelItemRange(BOOL bSelect, int nFirstItem, int nLastItem);
```

参数 bSelect 作为切换开关,对 nFirstItem 和 nLastItem 所定义的范围内的列表项进行选中或取消选中。

示　例:

```
CListBox * pList;
int nNumSel;
// create control ...
// add names to the list ...
nNumSel = pList- > SelectItemRange(TRUE, 0, 10);
```

语　法

函数 SetCurSel

函数 SetCurSel 声明如下:

```
int SetCurSel(int nSelect);
```

参数 nSelect 为新选中项的位置。如参数 nSelect 取值为- 1,则清除列表框的所有选择。如出错,函数返回 LB - ERR。

示　例:

```
CListBox* pList;
// create control ...
// add names to the list ...
```

```
pList->SetCurSel(-1); // Clear current selection
```

语　法

函数 SetSel

函数 SetSel 声明如下:

```
int SetSel(int nIndex, BOOL bSelect = TRUE);
```

参数 nIndex 指定要选中(当 bSelect 为 TRUE 时)或取消选中(当 bSelect 为 FALSE 时)的列表项。出错时,函数返回 LB_ERR。函数的结果仅标明选中/未选中的错误,你可以利用 SetSel 函数切换多项选择列表框中的多项的选中状态,一次一个。

示　例:

```
CListBox* pList;
// create control...
// add names to the list...
pList->SetSel(0, TRUE); // select first Item in first
```

语　法

函数 SetTopIndex

函数 SetTopIndex 声明如下:

```
int SetTopIndex(int nIndex);
```

参数 nIndex 为列表框中将成为第一个可视项的列表项的索引。该选择将滚动列表框,直至项 nIndex 成为第一可视项为止。若出错,函数返回 LB_ERR。其他情况下,返回值无意义。

示　例:

```
CListBox* pList;
int nIdxTop;
// create control...
// add names to the list...
nIdxTop = pList->SetTopIndex(10); // select 11th item
if(nIdxTop != 10)
MessageBox("Selection failed!", MB_OK, "Error!");
```

语　法

函数 Dir

函数 Dir 声明如下:

```
int Dir(UINT attr, const char FAR* lpWildCard);
```

参数 attr 指定属性的组合,属性列在表 18.2 中,这个表也列出了在头文件 DOS.H 中声明的等价的文件属性常量。参数 lpWildCard 指向文件名通配字符串,例如*.*,L*.EXE 或 A???.CPP。

示　例:

```
int nNumFiles;
CListBox* pList;
// create control...
// get all CTL*.CPP files
nNumFiles = pList->Dir(_A_ARCH, "CTL*.CPP");
```

表 18.2　成员函数 Dir 中 attr 参数的属性

属性值	头文件 DOS.H 中等价的常量	含　义
0x0000	_A_NORMAL	文件可用于输入和输出。
0x0001	_A_RDONLY	文件只读。
0x0002	_A_HIDDEN	文件隐含。
0x0004	_A_SYSTEM	为系统文件。
0x0010	_A_SUBDIR	参数 lpWildCard 确定的名字也可包含路径。
0x0020	_A_ARCH	文件档案位已设置。
0x4000		包括所有驱动器中与 lpWildCard 匹配的文件名。
0x8000		排他标志(禁止普通文件与指定文件在一起)。

列表框通知消息的响应

列表框控制发出各种类型的消息,如表 18.3 所示。这个表也列出了消息映像宏,它把各种命令和通知消息联系在一起。每种命令或通知消息都需要在控制的父窗口类中声明单独成员函数。

表 18.3　列表框通知消息

消　息	宏	含　义
WS_COMMAND	ON_COMMAND	Window 命令消息。
LBN_CHANGE	ON_LBN_SELCHANGE	表项用鼠标单击选中。
LBN_DBLCLK	ON_LBN_DBLCLK	表项用鼠标双击选中。
LBN_SETFOCUS	ON_LBN_SETFOCUS	列表框获得输入焦点。
LBN_KILLFOCUS	ON_LBN_KILLFOCUS	列表框失去输入焦点。
LBN_ERRSPACE	ON_LBN_ERRSPACE	列表框无法再分配动态内存来保存新表项。

列表操作测试程序

以下是一个简单的列表操作测试程序,它演示了怎样设置和查询普通的和选中的字符串以及怎样设定和查询单项选择列表框中的当前选项。本程序主要着重示范怎样使用本节前部分介绍的 CListBox 的大部分成员函数。程序中包括如下可体现测试特性的控制:

- 一个列表框控制。
- 一个允许输入和检索列表成员的 String Box 编辑控制。
- 一个能键入和检索当前选项位置的 IndexBox 编辑控制。
- 一个 Add String 下压按钮,用于把 String Box 的内容加入列表框(这个程序不允许加入重复的名字。如试着这样做,程序将显示一条警告消息)。
- 一个 Delete String 下压按钮,用于删除列表框中的当前选项(程序自动选中另一列表成员)。
- GetSelectedString 下压按钮,用于把当前列表选项拷贝到 String Box 中。
- SetSelectedString 下压按钮。用 String Box 中的串替换当前选中项。
- GetSelectedIndex 下压按钮,用于将当前选项的位置写到 Index Box 中。

· SetSelectedIndex 下压按钮，以 Index Box 中的整数值作为当前选项的位置。

· GetString 按钮将字符串(位置由出现在 IndexBox 中的值确定)拷贝到 String Box 中。

· Exit 下压按钮。

这些控制的使用涉及到操作一个排序列表框及其成员的各个方面。程序中保留当前选中项并禁止重名插入。

清单 18.1、18.2 和 18.3 中列出头文件 CTLLST1.H，资源描述文件 CTLLST1.RC 和程序文件 CTLLST1.CPP 的源代码。资源文件有一个单项菜单资源。为使程序简短，未使用加速键。

在\MSVC\VC21DAY 下建立子目录 CTLLST1，并把本项目所有的文件存到新目录下。.MAK 文件应包含文件 CTLLST1.CPP、CTLLST1.RC 和 CTLLST1.DEF。最后一个文件与第 14 天的表 14.2 中的 MINWINAP.DEF 相似。

编译并运行该程序。程序开始运行后，它会在列表框中放进一系列名字。可以使用各种下压按钮增加、删除或取得字符串。程序很直观、运行方便。

键　入　　清单 18.1　头文件 CTLLST1.H 的源代码

```
1: # define CM_EXIT (WM_USER + 100)
```

键　入　　清单 18.2　资源描述文件 CTLLST1.RC

```
1: # include < windows.h>
2: # include < afxres.h>
3: # include "ctllst1.h"
4:
5: EXITMENU MENU LOADONCALL MOVABLE PURE DISCARDABLE
6: BEGIN
7:     MENUITEM "E&xit", CM_EXIT
8: END
```

键　入　　清单 18.3　程序文件 CTLLST1.CPP 的源代码

```
1: # include < stdio.h>
2: # include < string.h>
3: # include < afxwin.h>
4: # include "ctllst1.h"
5:
6: // declare constants for the size and spacing of controls
7: const LowVertSpacing = 5;
8: const HiVertSpacing = 25;
9: const HorzSpacing = 50;
10: const Wctl = 150;
```

```
11: const Hctl = 30;
12: const Wbox = 2 * Wctl + HorzSpacing;
13: const Hlst = Hctl + LowVertSpacing + 4 * (Hctl + HiVertSpacing);
14: const MaxEditLen = 10;
15:
16: // declare the ID constants for the various control
17: # define ID_STRING_LST 101
18: # define ID_STRING_EDIT 102
19: # define ID_INDEX_EDIT 103
20: # define ID_ADDSTR_BTN 104
21: # define ID_DELSTR_BTN 105
22: # define ID_GETSELSTR_BTN 106
23: # define ID_SETSELSTR_BTN 107
24: # define ID_GETSELIDX_BTN 108
25: # define ID_SETSELIDX_BTN 109
26: # define ID_EXIT_BTN 110
27: # define ID_GETSTR_BTN 111
28:
29: class CxButton : public CButton
30: {
31: public:
32:
33:   BOOL Create(const char FAR* lpCaption, const RECT& rect,
34:               CWnd* pParentWnd, UINT nID)
35:   {
36:     return CButton::Create(lpCaption,
37:           WS_CHILD ©¦WS_VISIBLE ©¦BS_PUSHBUTTON,
38:           rect, pParentWnd, nID);
39:   }
40: };
41:
42: class CWindowApp: public CWinApp
43: {
44: public:
45:   virtual BOOL InitInstance();
46: };
47:
48: // expand the functionality of CFrameWnd by deriving
49: // class CAppWindow
50: class CAppWindow: public CFrameWnd
51: {
52: public:
53:
54:   CAppWindow();
55:   ~CAppWindow();
56:
57: protected:
58:   // handle creating the controls
59:   // declare the pointers to the controls
60:   CStatic* StringListTxt;
61:   CStatic* StringBoxTxt;
62:   CStatic* IndexBoxTxt;
63:   CListBox* StringLst;
64:   CEdit* StringBox;
```

```
65:   CEdit*  IndexBox;
66:   CxButton*  AddStrBtn;
67:   CxButton*  DelStrBtn;
68:   CxButton*  GetSelStrBtn;
69:   CxButton*  SetSelStrBtn;
70:   CxButton*  GetSelIdxBtn;
71:   CxButton*  SetSelIdxBtn;
72:   CxButton*  GetStrBtn;
73:   CxButton*  ExitBtn;
74:   char dataStr[MaxEditLen+ 1];   //   string for StringBox
75:   char indexStr[ MaxEditLen+ 1]; //   string for IndexBox
76:
77:   afx - msg int OnCreate(LPCREATESTRUCT lpCS);
78:   // initialize the String list box
79:   void InitStringLst();
80:   // handle adding a string to the list box
81:   afx - msg void HandleAddStrBtn();
82:   // handle deleting a string from the list box
83:   afx - msg void HandleDelStrBtn();
84:   // handle getting the selected text
85:   afx - msg void HandleGetSelStrBtn();
86:   // handle setting the selected text
87:   afx - msg void HandleSetSelStrBtn();
88:   // handle getting the index of the selected text
89:   afx - msg void HandleGetSelIdxBtn();
90:   // handle setting the index of the selected text
91:   afx - msg void HandleSetSelIdxBtn();
92:   // handle getting a string from the list box
93:   afx - msg void HandleGetStrBtn();
94:   // handle setting a string in the list box
95:   afx - msg void HandleExitBtn();
96:    { SendMessage (WM - CLOSE); }
97:   // handle exiting the application
98:   afx - msg void OnExit()
99:    { SendMessage(WM - CLOSE); }
100:   // handle closing the window
101:   afx - msg void OnClose();
102:   void makeRect(int nX, int nY, int nW, int nH, CRect& r):
103:
104:   // declare message map macro
105:   DECLARE - MESSAGE - MAP();
106: };
107:
108: CAppWindow::CAppWindow()
109: {
110:    // create the window
111:    Create(NULL, "Simple List Box Tester Application",
112:         WS - OVERLAPPEDWINDOW ©¦WS - MAXIMIZE,
113:         rectDefault, NULL, "EXITMENU");
114:    // insert the initial list of names
115:    InitStringLst();
116: }
117:
118: CAppWindow::~CAppWindow()
```

```
119: {
120:    delete StringListTxt;
121:    delete StringBoxTxt;
122:    delete IndexBoxTxt;
123:    delete StringLst;
124:    delete StringBox;
125:    delete IndexBox;
126:    delete AddStrBtn;
127:    delete DelStrBtn;
128:    delete GetSelStrBtn;
129:    delete SetSelStrBtn;
130:    delete GetSelIdxBtn;
131:    delete SetSelIdxBtn;
132:    delete GetStrBtn;
133:    delete ExitBtn;
134: }
135:
136: int CAppWindow::OnCreate(LPCREATESTRUCT lpCS)
137: {
138:     int nX0 = 30;
139:     int nX1 = nX0 + Wctl + HorzSpacing;
140:     int X2 = nX1 + Wctl + HorzSpacing;
141:     int nY0 = 50;
142:     int nX = nX0;
143:     int nY = nY0;
144:     int nY1;
145:     DWORD dwStaticStyle = WS_CHILD ©¦WS_VISIBLE ©¦SS_LEFT;
146:     DWORD dwEditStyle = WS_CHILD ©¦WS_VISIBLE ©¦ES_LEFT ©¦
147:                         WS_BORDER;
148:     DWORD dwScrollStyle = WS_CHILD ©¦WS_VISIBLE ©¦SS_VERT;
149:     DWORD dwListBoxStyle = WS_CHILD ©¦WS_VISIBLE ©¦ES_VSCROLL ©¦
150:                            LBS_STANDARD;
151:     CRect r;
152:
153:     // create the list box and its label
154:     makeRect(nX, nY, Wctl, Hctl, r);
155:     StringListTxt = new CStatic();
156:     StringListTxt->Create("List Box", dwStaticStyle, r, this);
157:     nY += Hctl + LowVertSpacing;
158:     makeRect(nX, nY, Wctl, Hlst, r);
159:     StringLst = new CListBox();
160:     StringLst->Create(dwListBoxStyle, r this, ID_STRING_LST);
161:
162:     // create the edit boxes and their labels
163:     nX = nX1;
164:     nY = nY0;
165:     makeRect(nX, nY, Wctl, Hctl, r);
166:     StringBoxTxt = new CStatic();
167:     StringBoxTxt->Create("StringBox", dwStaticStyle, r, this);
168:     nY += Hctl + LowVertSpacing;
169:     makeRect(nX, nY, Wctl, Hctl, r);
170:     StringBox = new CEdit();
171:     StringBox->Create(dwEditStyle, r this, ID_STRING_EDIT);
172:
```

```
173:     nY + = Hctl + HiVertSpacing;
174:     makeRect(nX, nY, Wctl, Hctl, r);
175:     IndexBoxTxt = new CStatic();
176:     IndexBoxTxt- > Create("Index Box", dwStaticStyle, r, this);
177:     nY + = Hctl + LowVertSpacing;
178:     makeRect(nX, nY, Wbox, Hctl, r);
179:     IndexBox = new CEdit();
180:     IndexBox- > Create(dwEditStyle, r this, ID - INDEX - EDIT);
181:
182:     // create the button controls
183:     nY + = Hctl + HiVertSpacing;
184:     nY1 = nY;
185:     makeRect(nX, nY, Wctl, Hctl, r);
186:     AddStrBtn = new CxButton();
187:     AddStrBtn- > Create("Add String", r, this, ID - ADDSTR - BTN);
188:
189:     nY + = Hctl + HiVertSpacing;
190:     makeRect(nX, nY, Wctl, Hctl, r);
191:     DelStrBtn = new CxButton();
192:     DelStrBtn- > Create("Delete String", r, this, ID - DELSTR - BTN);
193:
194:     nY + = Hctl + HiVertSpacing;
195:     makeRect(nX, nY, Wctl, Hctl, r);
196:     GetSelStrBtn = new CxButton();
197:     GetSelStrBtn- > Create("Get Selected String", r, this,
198:                            ID - GETSELSTR - BTN);
199:
200:     nY + = Hctl + HiVertSpacing;
201:     makeRect(nX, nY, Wctl, Hctl, r);
202:     SetSelStrBtn = new CxButton();
203:     SetSelStrBtn- > Create("Set Selected String", r, this,
204:                            ID - SETSELSTR - BTN);
205:
206:     // create the second row of buttons
207:     nY = nY1;
208:     nX = X2;
209:     makeRect(nX, nY, Wctl, Hctl, r);
210:     GetSelIdxBtn = new CxButton();
211:     GetSelIdxBtn- > Create("Get Selected Index", r, this,
212:                            ID - GETSELIDX - BTN);
213:
214:     nY + = Hctl + HiVertSpacing;
215:     makeRect(nX, nY, Wctl, Hctl, r);
216:     SetSelIdxBtn = new CxButton();
217:     SetSelIdxBtn- > Create("Set Selected Index", r, this,
218:                            ID - SETSELIDX - BTN);
219:
220:     nY + = Hctl + HiVertSpacing;
221:     makeRect(nX, nY, Wctl, Hctl, r);
222:     GetStrBtn = new CxButton();
223:     GetStrBtn- > Create("Get String by Index", r, this,
224:                            ID - GETSTR - BTN);
225:
226:     nY + = Hctl + HiVertSpacing;
```

```
227:     makeRect(nX, nY, Wbox, Hctl, r);
228:     ExitBtn = new CxButton();
229:     ExitBtn->Create("Exit", r, this, ID_EXIT_BTN);
230:
231:     return CFrameWnd::OnCreate(lpCS);
232:
233: }
234:
235: void CAppWindow::HandleAddStrBtn()
236: {
237:     int nIdx;
238:     // get the string in the String box
239:     StringBox->GetWindowText(dataStr, MaxEditLen);
240:     // exit if the string empty?
241:     if (dataStr[0] == '\0') return;
242:     // add the string if it is not already in the list box
243:     if (StringLst->FindString(-1, dataStr) < 0) {
244:       //   add the string and store the position of the new string
245:       nIdx = StringLst->AddString(dataStr);
246:       // make the added string the new selection
247:       StringLst->SetCurSel(nIdx);
248:     }
249:     else
250:       // handle the duplicate data error
251:       MessageBox("Cannot add duplicate names", "Bad Data", MB_OK);
252: }
253:
254: void CAppWindow::HandleDelStrBtn()
255: {
256:     // get the index of the currently selected list member
257:     int nIdx = StringLst->GetCurSel();
258:     // delete the currently selected list member
259:     StringLst->DeleteString(nIdx);
260:     // select another list member
261:     StringLst->SetCurSel((nIdx > 0) ? (nIdx-1) : 0);
262: }
263:
264: void CAppWindow::HandleGetSelStrBtn()
265: {
266:     // get the selected list item
267:     StringLst->GetText(StringLst->GetCurSel(), dataStr);
268:     // store it in the String box
269:     StringBox->SetWindowText(dataStr);
270: }
271:
272: void CAppWindow::HandleSetSelStrBtn()
273: {
274:     // get the index of the currently selected list member
275:     int nIdx = StringLst->GetCurSel();
276:     // get the string to replace the currently selected
277:     // list item
278:     StringBox->GetWindowText(dataStr, MaxEditLen);
279:     // is the candidate string not in the list?
280:     if (StringLst->FindString(-1, dataStr) < 0) {
```

```
281:        // delete the current selection
282:        StringLst->DeleteString(nIdx);
283:        // insert the new selection
284:        nIdx = StringLst->AddString(dataStr);
285:        // select the inserted string
286:        StringLst->SetCurSel(nIdx);
287:     }
288:     else
289:        MessageBox("Cannot add duplicate names", "Bad Data", MB_OK);
290: }
291:
292: void CAppWindow::HandleGetSelIdxBtn()
293: {
294:     sprintf(indexStr, "%d", StringLst->GetCurSel());
295:     IndexBox->SetWindowText(indexStr);
296: }
297:
298: void CAppWindow::HandleSetSelIdxBtn()
299: {
300:     IndexBox->GetWindowText(indexStr, MaxEditLen);
301:     StringLst->SetCurSel(atoi(indexStr));
302: }
303:
304: void CAppWindow::HandleGetStrBtn()
305: {
306:     int nIdx;
307:     // get the index from the Index box
308:     IndexBox->GetWindowText(indexStr, MaxEditLen);
309:     nIdx = atoi(indexStr);
310:     // get the target string from the list box
311:     StringLst->GetText(nIdx, dataStr);
312:     // write the list member in the String box
313:     StringBox->SetWindowText(dataStr);
314: }
315:
316: void CAppWindow::OnClose()
317: {
318:     if (MessageBox("Want to close this application",
319:           "Query", MB_YESNO |MB_ICONQUESTION) == IDYES)
320:        DestroyWindow();
321: }
322:
323: void CAppWindow::InitStringLst()
324: {
325:     // add data in the list box
326:     StringLst->AddString("Edward");
327:     StringLst->AddString("John");
328:     StringLst->AddString("Anne");
329:     StringLst->AddString("Elaine");
330:     StringLst->AddString("Joseph");
331:     StringLst->AddString("Clement");
332:     StringLst->AddString("Daisy");
333:     StringLst->AddString("Tony");
334:     // select the second item
```

```
335:     StringLst- > SetCurSel(1);
336: }
337:
338: void CAppWindow::makeRect(int nX, int nY, int nW, int nH,
339:                            CRect& r)
340: {
341:     r.top = nY;
342:     r.left = nX;
343:     r.bottom = nY + nH;
344:     r.right = nX + nW;
345: }
346:
347: BEGIN_MESSAGE_MAP(CAppWindow, CFrameWnd)
348:   ON_WM_CREATE()
349:   ON_BN_CLICKED(ID_ADDSTR_BTN, HandleAddStrBtn)
350:   ON_BN_CLICKED(ID_DELSTR_BTN, HandleDelStrBtn)
351:   ON_BN_CLICKED(ID_GETSELSTR_BTN, HandleGetSelStrBtn)
352:   ON_BN_CLICKED(ID_SETSELSTR_BTN, HandleSetSelStrBtn)
353:   ON_BN_CLICKED(ID_GETSELIDX_BTN, HandleGetSelIdxBtn)
354:   ON_BN_CLICKED(ID_SETSELIDX_BTN, HandleSetSelIdxBtn)
355:   ON_BN_CLICKED(ID_GETSTR_BTN, HandleGetStrBtn)
356:   ON_BN_CLICKED(ID_EXIT_BTN, HandleExitBtn)
357:   ON_COMMAND(CM_EXIT, OnExit)
358:   ON_WM_CLOSE()
359: END_MESSAGE_MAP()
360:
361: // Construct the CWindowApp's m_pMainWnd data member
362: BOOL CWindowApp::InitInstance()
363: {
364:   m_pMainWnd = new CAppWindow()
365:   m_pMainWnd- > ShowWindow(m_nCmdShow);
366:   m_pMainWnd- > UpdateWindow();
367:   return TRUE;
368: }
369:
370: // application's constructor initializes and runs the app
371: CWindowApp WindowApp;
```

图 18.1 中显示了应用程序 CTLLST1.EXE 的一个运行范例。

分　析

清单 18.3 的程序中声明了两组常量。第 7 到 14 行为第 1 组，指定了使用的各种控制的位置和大小；第 17 到 27 行为第 2 组，用来指定各种控制的 ID。

程序声明了应用程序类 CWindowApp、窗口类 CAppWindow 和按钮类 CxButton。类 CAppWindow 在第 60 到 73 行声明了若干数据成员，它们是指向主窗口所属的控制的指针。此类还声明了两个用于处理 StringBox 和 IndexBox 编辑控制的字符串。CAppWindow 类也声明了一个构造函数，一个析构函数和若干用于响应各种下压按钮控制所发出的通知消息的成员函数。

CAppWindow 构造函数执行两类任务：

图 18.1　应用程序 CTLLST1.EXE 的运行范例

· 调用成员函数 Create, 建立应用程序窗口。
· 调用成员函数 InitStringLst, 用数据初始化 Names 列表框。

第 136 到 233 行定义了成员函数 OnCreate, 它用适当的风格建立控制。例如, 列表框控制具有一个竖直滚动条并自动维护有序的字符串列表。

第 235 到 252 行定义了成员函数 HandleAddStrBtn, 该函数把 String Box 中的字符串加入到列表框控制中。函数执行以下任务:

· 获得 String Box 编辑控制中的字符串, 并存储到 dataStr 数据成员中。
· 在 dataStr 成员存储一个空字符串时退出。
· 检验加入的字符串是否已存在列表框中, 使用函数 FindString 检测是否企图加入重复的字符串(如果 FindString 返回一个负值, HandleAddStrBtn 继续往下执行, 如不返回一个负值, HandleAddStrBtn 显示一条消息通知用户, 不能在列表框中加入重复字符串)。
· 用 AddString 成员函数把 dataStr 成员的字符串加入到列表框中, 把该字符串的位置赋给局域变量 nIdx。
· 调用 SetCurSel 函数(参数为 nIdx), 把加入的字符串变成当前选中项。

第 254 到 262 行中定义了成员函数 HandleDelStrBtn, 该函数删除当前选项, 具体操作如下:

· 调用函数 GetCurSel 获得当前选中项的位置, 并把当前选项的位置存储在局部变量 nIdx 中。
· 以 nIdx 为参数, 调用 DeleteString 函数删除选中项。

· 选择 nIdx- 1 位置处的列表项作为新选中项(如果变量 nIdx 为 0, 新的第一列表项变成新的选中项)。

第 264 到 270 行定义了成员函数 HandleGetSelStrBtn, 该函数拷贝当前选中项到 String Box 编辑控制中。函数执行如下任务:

· 调用 GetText 和 GetCurSel 函数, 把当前选中项拷贝到 dataStr 成员中。
· 用 dataStr 成员的字符重写 String Box 的内容。

第 272 到 290 行定义了成员函数 HandleSetSelStrBtn, 该函数用 String Box 编辑控制中的字符串重写当前选中项。因列表要维持表项的有序性, 因此替代串的位置与原选中项的可能有所不同。本函数执行如下的任务:

· 调用 GetCurSel 函数获得当前选中项的位置, 并把值赋给局域变量 nIdx。
· 把 String Box 中的文本拷贝到 dataStr 成员中。
· 用函数 FindString 检验 dataStr 中的字符串是否已存在于列表框中(如 dataStr 中的字符串对于列表是新的, 函数使用 DeleteString 函数删除当前选中项, 使用 AddString 函数增加 dataStr, 然后调用 SetCurSel 函数选中加入的字符串)。

如果 dataStr 中的字符串与一个列表项匹配, 本函数显示一条消息提示用户: 列表框中不能加入重复的字符串。如果试图用同样的字符串重写当前选中项, 警告也会出现。

第 298 到 302 行定义了成员函数 HandleSetSelIdxBtn, 它读出 Index Box 编辑控制的值, 并使用这个值去设定新的当前选项。本函数使用 SetCurSel 函数设置新选项。

第 292 到 296 行定义了成员函数 HandleGetIdxBtn, 它把当前选项的位置写进 Index Box 编辑框。本函数使用 GetCurSel 函数获得找到的位置。

第 304 到 314 行定义了成员函数 HandleGetStrBtn, 它允许检索其位置出现在 IndexBox 编辑控制中的列表项。本函数执行如下的任务:

· 把 IndexBox 中的字符拷贝到 indexStr 数据成员中。
· 把 IndexStr 中的字符串转换成整型, 放到局部变量 nIdx 中。
· 把 nIdx 位置处的列表项中的字符拷进 dataStr 成员。
· 把 dataStr 中的字符写进 String Box 编辑控制中。

18.2　处理多项选择列表

这一节将演示多重选择列表的使用, 主要集中讨论获得和设置选中的字符串和它们的索引。有两种方式可以在一个列表框中进行多重选择, 这些方式都依赖于在建立一个 CListBox 实例时是否了设置 LBS_EXTENDEDSEL 风格。如设置本风格, 便可通过按住 Shift 并单击鼠标来快速扩展选中项的范围。本风格的不利之处在于, 你只能在列表框中选取连续的选中块(使用鼠标或光标键)。使用 SetSel 或 SetItemRange 成员函数可以使程序选中不连续的项。当然, 这种方法要求应用程序用户作出额外的努力并需要几个额外的控制。与此相反, 如没有设置 LBS_EXTENDEDSEL 风格, 则可以用鼠标分别击中想选的项, 这样很容易得到分散的选项。这种选择方式的缺点是, 必须击中每个要选的项, 即

使它们是相邻的。用户可以选择最适合自己的 MFC 应用程序用户界面的选择模式。

多项选择列表测试

图 18.2 显示了 CTLLST2.EXE 应用程序的一个运行范例——它用于演示如何在一个列表框中设置和查询多个选择项——也展示应用程序使用的各种控制。测试程序使用的控制和它们支持的操作如下:

- 一个具有 LBS_MULIPLESEL 风格,但不具有 LBS_EXTENDEDSEL 风格的多重选择列表(列表框中的项按升序排列)。
- 一个标名为 Names Box 的多行编辑框控制,其中包含选中字符串的名字。
- 一个标题为 GetNames 下压按钮,用于把列表框选项拷进 Names Box 编辑控制(每个选中项在编辑框中都单独占一行,每次击中按钮时,Names Box 编辑控制中的原有内容就会丢失)。
- 一个标题为 SetNames 的下压按钮,用于选中或取消选中那些与 Names Box 编辑控制中匹配的列表框项(匹配无需严格,但 Names Box 的字符串必须与列表项的起头字符匹配)。
- 一个标名为 Indices Box 的多行编辑框控制,它包含选中字符串的索引。
- 一个标题为 Get Indices 的下压按钮,它将列表框选中项的索引拷贝到 Indices Box 编辑控制中(每个选项的索引在编辑框中单独占一行,每次单击此按钮时,Indices Box 编辑控制的原有内容就会丢失)。
- 一个标题为 SetIndices 的下压按钮,用于选中或取消选中其索引在 Indices Box 编辑控制中出现的列表框项。
- 一个标题为 Deselect Mode 的选中框控制,用于确定 SetNames 和 SetIndices 按钮是否选中或取消选中列表项。
- 用于标名列表框和编辑控制的静态文本控制。

这个多重选择列表测试应用程序初步地操练了 GetSel, SetSel, AddString, DeleteString 等类 CListbox 的成员函数。清单 18.4、18.5 和 18.6 分别包含 CTLLST2.H, CTLLST2.RC 和 CTLLST2.CPP 文件的源代码。

在\MSVC\VC21DAY 下建立子目录 CTLLST2,并把项目的所有文件存在新目录下。.MAK 文件应包含文件 CTLLST2.CPP, CTLLST2.RC 和 CTLLST2.DEF,最后一个文件与第 14 天的清单 14.2 中的 MINWINAP.DEF 相似。

编译并运行本程序。应用程序用许多名字初始化列表框,这个特性可节省你的时间,不用你亲自输入名字。选中几个列表项,并单击 GetNames 和 GetIndices 按钮控制。选中的字符串和索引分别出现在 Names Box 和 Indices Box 编辑控制中。现在编辑 Names Box 的行,在每行上删除尾部的一个或两个字符,建立部分名。单击 SetNames 下压按钮。你将看见什么?因为 Names Box 的字符串仍与相同的列表框选项匹配,这个选项未改变。在 Indices Box 中键入一组新索引,并单击 SetIndices 下压按钮,程序现在在列表框中显示一组新的选中项。要产生新的选中项,也可以删除 Names Box 的文本,然后键入列表框中能发现的名字,再单击 Set Names 按钮,查看新的列表框选项。当完成程序实验时,点中

图 18.2　CTLLST2.EXE 应用程序的一个运行范例

Exit 菜单项或使用 Alt-X 键退出。

键　入　　清单 18.4　头文件 CTLLST2.H 源代码

```
1: #define CM_EXIT (WM_USER + 100)
```

键　入　　清单 18.5　资源描述文件 CTLLST2.RC

```
1: #include <windows.h>
2: #include <afxres.h>
3: #include "ctllst2.h"
4:
5: EXITMENU MENU LOADONCALL MOVEABLE PURE DISCARDABLE
6: BEGIN
7:     MENUITEM "E&xit", CM_EXIT
8: END
```

键　入　　清单 18.6　程序文件 CTLLIST2.CPP 的源代码

```
1: #include <stdio.h>
2: #include <string.h>
3: #include <afxwin.h>
4: #include "ctllst2.h"
5:
6: // declare the constants for the dimensions and spacing
```

```
 7: // of the various controls
 8: const Wtxt = 100;
 9: const Htxt = 20;
10: const TxtVertSpacing = 50;
11: const Wlst = 200;
12: const Hlst = 300;
13: const LstHorzSpacing = 50;
14: const LstVertSpacing = 20;
15: const Wbox = 250;
16: const Hbox = 100;
17: const BoxVertSpacing = 20;
18: const Wbtn = 100;
19: const Hbtn = 30;
20: const BtnHorzSpacing = 50;
21: const BtnVertSpacing = 10;
22: const Wchk = 200;
23: const Hchk = 20;
24: // miscellaneous constants
25: const MaxString = 40;
26: const MaxSelections = 35; // maximum number of selections
27: // declare the ID constants for the various controls
28: # define ID_NAMES_LST 101
29: # define ID_NAMES_EDIT 102
30: # define ID_INDICES_EDIT 103
31: # define ID_GETNAMES_BTN 104
32: # define ID_SETNAMES_BTN 105
33: # define ID_GETINDICES_BTN 106
34: # define ID_SETINDICES_BTN 107
35: # define ID_SELECT_CHK 108
36: // declare constants for the check box and radio button states
37: const BF_CHECKED = 1;
38: const BF_UNCHECKED = 0;
39:
40: // declare extended edit controls class
41: class CxEdit： public CEdit
42: {
43: public:
44:
45:    BOOL GetLine(LPSTR lpString, int nStrSize, int nLineNumber);
46:    int GetLineLength(int nLineNumber);
47:    int GetNumOfLines();
48: }
49:
50: // declare the extended button class
51: class CxButton： public CButton
52: {
53: public
54:
55:    BOOL Create(const char FAR* lpCaption, const RECT& rect,
56:                CWnd* pParentWnd, UINT nID)
57:    {
58:          return CButton::Create(lpCaption,
59:                         WS_CHILD ©¦WS_VISIBLE ©¦
60:                         WS_TABSTOP ©¦BS_PUSHBUTTON,
```

```
61:                         rect, pParentWnd, nID);
62:    }
63: };
64:
65: // declare the check box class
66: class CCheckBox:  public CButton
67: {
68: public
69:    BOOL Create(const char FAR* lpCaption, const RECT& rect,
70:                CWnd* pParentWnd, UINT nID);
71:    {
72:       return CButton::Create(lpCaption,
73:                         WS_CHILD ©¦WS_VISIBLE ©¦
74:                         WS_TABSTOP ©¦BS_AUTOCHECKBOX,
75:                         rect, pParentWnd, nID);
76:    }
77:
78: };
79:
80:
81: // expand the functionality of CFrameWnd by deriving
82: // class CAppWindow
83: class CAppWindow:  public CFrameWnd
84: {
85: public
86:
87:    CAppWindow();
88:    ~CAppWindow();
89:
90: protected:
91:    // declare pointers to controls
92:    CStatic* NamesListTxt;
93:    CStatic* NamesBoxTxt;
94:    CStatic* IndicesBoxTxt;
95:    CListBox* NamesLst;
96:    CxEdit* NamesBox;
97:    CxEdit* IndicesBox;
98:    CxButton* GetNamesBtn;
99:    CxButton* SetNamesBtn;
100:    CxButton* GetIndicesBtn;
101:    CxButton* SetIndicesBtn;
102:    CCheckBox* SelectChk;
103:    // declare edit control insertion buffer
104:    CString insertBuffer;
105:    // array of string pointers
106:    LPSTR names[MaxSelections];
107:    // array of integers to store the selection indices
108:    int indices[MaxSelections]
109:
110:    // handle creating the list box
111:    afx_msg int OnCreate(LPCREATESTRUCT lpCS);
112:    // initialize the list box
113:    void InitNamesLst();
114:    // handle getting the selections
```

```
115:    afx - msg void HandleGetNamesBtn();
116:    // handle setting the selections
117:    afx - msg void HandleSetNamesBtn();
118:    // handle getting the selection indices
119:    afx - msg void HandleGetIndicesBtn();
120:    // handle setting the selection indices
121:    afx - msg void HandleSetIndicesBtn();
122:    // handle exiting the application
123:    afx - msg void OnExit()
124:    { SendMessage(WM - CLOSE); }
125:    // handle closing the window
126:    virtual void OnClose();
127:    void resetSelections();
128:    void makeRect(int nX, int nY, int nW, int nH, CRect& r);
129:
130:    // declare message map macro
131:    DECLARE - MESSAGE - MAP();
132: };
133:
134: class CWindowApp:  public CWinApp
135: {
136: public:
137:       virtual BOOL InitInstance();
138: };
139:
140: BOOL CxEdit::GetLine(LPSTR lpString, int nStrSize,
141:                          int nLineNumber)
142: {
143:    int nCopyCount;
144:    BOOL bResult;
145:
146:    if (nStrSize < = 0)
147:       return FALSE;
148:    bResult = (nStrSize > = GetLineLength(nLineNumber) + 1) ? TRUE
149:                                                            : FALSE;
150:    if (nStrSize = = 1) {
151:       lpString[0] = '\0';
152:       return bResult;
153:    }
154:    ((WORD FAR * )lpString([0] = nStrSize;
155:    nCopyCount = (WORD)(SendMessage(EM - GETLINE, nLineNumber,
156:                                        long(lpString)));
157:    if (nCopyCount) {
158:       // Windows returns non- null terminated string
159:       lpString[nCopyCount] = '\0';
160:       return bResult;
161:    }
162:    else
163:       return FALSE;
164: }
165:
166: int CxEdit::GetLineLength(int nLineNumber)
167: {
168:    int nStartPos = - 1;
```

```
169:
170:    if (nLineNumber > -1)
171:       nStartPos = LineIndex(nLineNumber);
172:    return (WORD)SendMessage(EM_LINELENGTH, nStartPos);
173: }
174:
175: int CxEdit::GetNumOfLines()
176: {
177:    // get the size of the text in the edit control
178:    int nSize = GetWindowTextLength();
179:    // initialize the function's result
180:    int nCount = (nSize == 0) ? 0 : 1;
181:    // return 0 if the edit box is empty
182:    if (nSize == 0)
183:       return 0;
184:    // create a string object of the same nSize
185:    CString s('', nSize);
186:    // copy the text in the edit control to the string object
187:    GetWindowText(s.GetBuffer(nSize), nSize);
188:    s.ReleaseBuffer(); // release any extra space
189:    // loop to nCount the number of lines by counting the number
190:    // of '\n' characters (assuming they only occur at a line
191:    // break;
192:    for (int i = 0; i < s.GetLength();)
193:       if (s[i++] == '\r' && s[i] == '\n') {
194:          nCount++;
195:          i++;
196:       }
197:    return nCount;
198: }
199:
200: CAppWindow::CAppWindow()
201: {
202:    // Create the window
203:    Create(NULL, "Multiple-Selection List Tester",
204:          WS_OVERLAPPEDWINDOW ©¦WS_MAXLINE,
205:          rectDefault, NULL, "EXITMENU");
206:    // insert the initial names in the lists
207:    InitNamesList();
208: }
209:
210: CAppWindow::~CAppWindow()
211: {
212:    delete NamesListTxt;
213:    delete NamesBoxTxt;
214:    delete IndicesBoxTxt;
215:    delete NamesLst;
216:    delete NamesBox;
217:    delete IndicesBox;
218:    delete GetNamesBtn;
219:    delete SetNamesBtn;
220:    delete GetIndicesBtn;
221:    delete SetIndicesBtn;
222:    delete SelectChk;
```

```
223:    // deallocate the dynamic string space
224:    for (int i = 0; i < MaxSelections; i++)
225:        delete names[i];
226: }
227:
228: int CAppWindow::OnCreate(LPCREATESTRUCT lpCS)
229: {
230:    int nX0 = 100;
231:    int nY0 = 50;
232:    int nX = nX0;
233:    int nY = nY0;
234:    DWORD dwStaticStyle = WS_CHILD|WS_VISIBLE|SS_LEFT;
235:    DWORD dwEditStyle = WS_CHILD|WS_VSCROLL|ES|LEFT|
236:                        WS_BORDER|WS_VISIBLE|WS_TABSTOP|
237:                        ES_AUTOHSCROLL|ES_AUTOVSCROLL|
238:                        ES_MULTILINE;
239:    DWORD dwListBoxStyle = WS_CHILD|WS_VISIBLE|LBS_STANDARD|
240:                           LBS_MULTIPLESEL;
241:    CRect r;
242:
243:    // create the name list box and its label
244:    makeRect(nX, nY, Wtxt, Htxt, r);
245:    NamesListTxt = new CStatic();
246:    NamesListTxt->Create("Names List", dwStaticStyle, r, this);
247:    nY += Htxt + TxtVertSpacing;
248:    makeRect(nX, nY, Wlst, Hlst, r);
249:    NamesLst = new CListBox();
250:    NamesLst->Create(dwListBoxStyle, r, this, ID_NAMES_LST);
251:    // create the Deselect Mode check box
252:    nY += Hlst + LstVertSpacing;
253:    makeRect(nX, nY, Wchk, Hchk, r);
254:    SelectChk = new CCheckBox();
255:    SelectChk->Create("Deselect Mode", r, this, ID_SELECT_CHK);
256:    nX0 += Wlst + LstHorzSpacing;
257:    nX = nX0;
258:    nY = nY0;
259:    // creates the multiline Names Box edit control and its label
260:    makeRect(nX, nY, Wtxt, Htxt, r);
261:    NamesBoxTxt = new CStatic();
262:    NamesBoxTxt->Create("Names Box", dwStaticStyle, r, this);
263:    nY += Htxt + TxtVertSpacing;
264:    makeRect(nX, nY, Wbox, Hbox, r);
265:    NamesBox = new CxEdit();
266:    NamesBox->Create(dwEditStyle, r, this, ID_NAMES_EDIT);
267:    // create the Get Names button
268:    nY += Hbox + BoxVertSpacing;
269:    makeRect(nX, nY, Wbtn, Hbtn, r);
270:    GetNamesBtn = new CxButton();
271:    GetNamesBtn->Create("Get Names", r, this, ID_GETNAMES_BTN);
272:    // create the Set Names button
273:    nX += Wbtn + BtnHorzSpacing;
274:    makeRect(nX, nY, Wbtn, Hbtn, r);
275:    SetNamesBtn = new CxButton();
276:    SetNamesBtn->Create("Set Names", r, this, ID_SETNAMES_BTN);
```

```
277:    nX = nX0;
278:    nY += Hbtn + BtnVertSpacing;
279:    // create the multiline Indices Box edit control and its label
280:    makeRect(nX, nY, Wtxt, Htxt, r);
281:    IndicesBoxTxt = new CStatic();
282:    IndicesBoxTxt->Create("Indices Box", dwStaticStyle, r, this);
283:    nY += Htxt + TxtVertSpacing;
284:    makeRect(nX, nY, Wbox, Hbox, r);
285:    IndicesBox = new CxEdit();
286:    IndicesBox->Create(dwEditStyle, r, this, ID_INDICES_EDIT);
287:    // create the GetIndices button
288:    nY += Hbox + BoxVertSpacing;
289:    makeRect(nX, nY, Wbtn, Hbtn, r);
290:    GetIndicesBtn = new CxButton();
291:    GeIndicesBtn->Create("Get Indices", r, this,
292:                         ID_GETINDICES_BTN);
293:    // create the Set Indices button
294:    nX += Wbtn + BtnHorzSpacing;
295:    makeRect(nX, nY, Wbtn, Hbtn, r);
296:    SetIndicesBtn = new CxButton();
297:    SetIndicesBtn->Create("Set Indices", r, this
298:                          ID_SETINDICES_BTN);
299:    // allocate the dynamic space for the array of strings accessed
300:    // by the array of pointers names
301:    for (int i = 0; i < MaxSelections; i++)
302:      names[i] = new char[MaxString + 1];
303:    return CFrameWnd::OnCreate(lpCS);
304: }
305:
306: void CAppWindow::InitNamesLst()
307: {
308:    char * p[] = { "John", "Ribert", "Melody", "Charles", "Olivia",
309:                   "Richard", "James", "Anne", "Keith", "Brian",
310:                   "Lisa", "Margie", "Thomas", "Joseph", "Donald",
311:                   "Bert", "George", "Ronald", "Katie", "Susan",
312:                   "Joyce", "David", "Paul", "Mark", "Luke",
313:                   "Amos", "Matthew", "Mary", "Patrick", "Clement",
314:                   "Daisy", "EOL" };
315:    for (int i = 0; p[i] != "EOL"; i++)
316:      NamesLst->AddString(p[i]);
317: }
318:
319: void CAppWindow::HandleGetNamesBtn()
320: {
321:    char s[MaxString+ 1];
322:    // get the indices of the selected strings
323:    int n = NamesLst->GetSelItems(MaxSelections, indices);
324:
325:    // exit if n is not positive
326:    if (n <= 0) return;
327:    // clear the insertion buffer
328:    insertBuffer = "";
329:    //  concatenate the selected strings in the insertion buffer
330:    for (int i = 0; i < n; i++) {
```

```
331:       NamesLst->GetText(indices[i], names[i]);
332:       sprintf(s, "%s\r\n", names[i]);
333:       insertBuffer += s;
334:    }
335:    // copy the insertion buffer to the Names Box edit control
336:    NamesBox->SetWindowText((const char *)insertBuffer);
337: }
338:
339: void CAppWindow::HandleSetNamesBtn()
340: {
341:    char s[MaxString+1];
342:    int j;
343:    // get the number of lines in the Names Box edit control
344:    int n = NamesBox->GetNumOfLines();
345:    // get the select status from the Deselect Mode check box
346:    BOOL bShouldSet =
347:       (SelectChk->GetCheck() == BF_UNCHECKED) ? TRUE : FALSE;
348:
349:    resetSelections();
350:    // read the lines from the Names Box control
351:    for (int i = 0; i < n; i++)
352:       // get the ith name from the Names Box edit control
353:       NamesBox->GetLine(s, MaxString, i);
354:       // find its match in the list box
355:       j = NamesLst->FindString(-1, s);
356:       // select the matching element
357:       if (j != LB_ERR)
358:          NamesLst->SetSel(j, bShouldSet);
359:    }
360: }
361:
362: void CAppWindow::HandleGetIndicesBtn()
363: {
364:    char s[MaxString + 1];
365:    // get the selected indices
366:    int n = NamesLst->GetSelItems(MaxSelections, indices);
367:
368:    // exit if n is not positive
369:    if (n <= 0) return;
370:    // initialize the insertion buffer
371:    insertBuffer = "";
372:    // concatenate the selected indices in the insertion buffer
373:    for (int i = 0; i < n; i++) {
374:       sprintf(s, "%d\r\n", indices[i]);
375:       insertBuffer += s;
376:    }
377:    // copy the insertion buffer to the Names edit control
378:    IndicesBox->SetWindowText((const char *) insertBuffer);
379: }
380:
381: void CAppWindow::HandleSetIndicesBtn()
382: {
383:    char s[MaxString + 1];
384:    int j;
```

```
385:
386:    resetSelections();
387:    // get the number of lines in the Indices Box edit control
388:    int n = IndicesBox->GetNumOfLines();
389:    // get the select status from the Deselect Mode check box
390:    BOOL bShouldSet =
391:       (SelectChk->GetCheck() == BF_UNCHECKED) ? TRUE : FALSE;
392:
393:    // read the lines from the Names Box control and set the
394:    // corresponding names in the list box
395:    for (int i = 0; i < n; i++) {
396:       IndicesBox->GetLine(s, MaxString, i);
397:       j = atoi(s);
398:       NamesLst->SetSel(j, bShouldSet);
399:    }
400: }
401:
402: void CAppWindow::OnClose()
403: {
404:    if (MessageBox("Want to close this application",
405:                   "Query", MB_YESNO ¦MB_ICONQUESTION) == IDYES)
406:       DestroyWindow();
407: }
408:
409: void CAppWindow::resetSelections()
410: {
411:    int n = NamesLst->GetSelItems(MaxSelections, indices);
412:
413:    for (int i = 0; i < n; i++)
414:          NamesLst->SetSel(indices[i], FALSE);
415: }
416:
417: void CAppWindow::makeRect(int nX, nY, int nW, int nH,
418:                           CRect& r)
419: {
420:    r.top = nY;
421:    r.left = nX;
422:    r.bottom = nY + nH;
423:    r.right = nX + nW;
424: }
425:
426: BEGIN_MESSAGE_MAP(CAppWindow, CFrameWnd)
427:    ON_WM_CREATE()
428:    ON_BN_CLICKED(ID_GETNAMES_BTN, HandleGetNamesBtn)
429:    ON_BN_CLICKED(ID_SETNAMES_BTN, HandleSetNamesBtn)
430:    ON_BN_CLICKED(ID_GETINDICES_BTN, HandleGetIndicesBtn)
431:    ON_BN_CLICKED(ID_SETINDICES_BTN, HandleSetIndicesBtn)
432:    ON_COMMAND(CM_EXIT, OnExit)
433:    ON_WM_CLOSE()
434: END_MESSAGE_MAP()
435:
436: // Construct the CWindowApp's m_pMainWnd data member
437: BOOL CWindowApp::InitInstance()
438: {
```

```
439:    m_pMainWnd = new CAppWindow();
440:    m_pMainWnd->ShowWindow(m_nCmdShow);
441:    m_pMainWnd->UpdateWindow();
442:    return TRUE;
443: }
444:
445: // application's constructor initializes and runs the app
446: CWindowApp WindowApp;;
```

分 析

清单 18.6 中的程序声明了如下的几组常量:

- 第 8 到 23 行定义的常量指定各种应用程序控制的尺寸和间距。
- 第 25 到 26 行的常量声明字符串的最大长度和选项数的最大值。
- 第 28 到 35 行的常量建立各种控制的 ID。
- 第 37 到 38 行的常量声明选中框状态常量,即 BF_CHECKED 和 BF_UNCHECKED。

测试程序声明了应用程序类 CWindowApp、窗口类 CAppWindow,编辑控制类 CxEdit,按钮类 CxButton 和选中框类 CCheckBox。最后三个类用和以往程序相似的方式扩展了 CEdit 和 CButton 类。

类 CxEdit 引入了一个附加的成员函数 GetNumOfLines,该函数用于计算一个编辑控制中的行数。当应用程序分别读一个多行编辑控制的单独行时,返回的结果就有用。

CAppWindow 类定义了两组数据成员,第一组提供指向应用程序中使用的各种控制的指针;第二组数据成员包含字符串指针数组 Names 以及整数数组 Indices,这两个数组都有 MaxSelections 个成员。

CAppWindow 类声明了一个类构造函数、一个析构函数和一个成员函数集。析构函数删除动态控制实例和动态字符串。

CAppWindow 构造函数引用 Create 函数和调用 InitNamesLst 函数初始化 Names 列表框。

第 228 到 304 行定义了成员函数 OnCreate,它用于建立各种控制并建立动态字符串数组。本函数给 CListbox 实例设置包括 LBS_MULTIPLESEL 风格在内的风格。如果你要使用程序在扩展列表框选中项时可用 Shift 键,在初始化局部变量 dwListBoxStyle 的表达式中加入LBS_EXTENDEDSEL。

第 319 到 337 行定义了成员函数 HandleGetNamesBtn,此函数响应击中 GetName 压按钮时发出的通知消息。本函数执行如下的任务:

- 调用 GetSelItems 函数通过将列表选择项拷贝进用 Indices 指针数组存取的列表框选项的索引数组中来获得列表选择,且至多能拷贝 MaxSelections 个索引(本函数的结果赋给局域变量 n)。
- 如果变量 n 为负,退出函数。
- 初始化插入缓存区 insertBuffer。

· 使用一个循环把名字数组存取的字符串添加进插入缓冲区 insertBuffer。
· 将插入缓冲区的内容拷贝到 Names Box 控制中。

第 339 到 360 行定义了成员函数 HandleSetNamesBtn, 响应 SetNames 下压按钮控制的通知消息。该函数执行如下任务:

· 获得 Names Box 控制中的行数。
· 获得 Deselect Mode 选中框的选中状态, 并将等价于选中状态的一个布尔值赋给逻辑变量 bShouldSet。
· 调用 resetSelections 成员函数取消选中当前选中项。
· 把 Names Box 控制的行拷进指针名数组存取的字符串数组。
· 使用一个循环去读出 Names Box 控制的每一行, 随后选中这些行(这涉及到使用 FindString。函数定位匹配的列表框项的索引, 接着用 SetSel 函数根据局域变量 shouldSet 的值选中或不选中匹配元素)。

第 386 到 379 行定义了成员函数 HandleGetIndicesBtn, 它响应 GetIndices 下压按钮控制发出的通知消息。该函数执行如下的任务:

· 调用 GetSelItems 函数获得列表框选项, 将这些选项拷贝到整型索引数组中, 并指定至多达 MaxSelections 个索引(本函数结果赋给局部变量 n)。
· 如果变量 n 为负值, 退出本函数。
· 用空字符串初始化插入缓存区 insertBuffer。
· 使用一个循环, 把选中项的索引转换为字符串映像, 并加入到插入缓存区。
· 将插入缓存区的字符串拷贝到 Indices Box 编辑控制中。

第 381 到 400 行定义了成员函数 HandleSetIndiceBtn, 响应 SetIndices 下压按钮控制发出的通知消息。本函数执行如下的任务:

· 获得 Indices Box 控制的行数(使用 CxEdit∷GetNumOfLines() 成员函数)。
· 获得 Deselect Mode 选中框的选中状态, 将等价于选中状态的布尔型值赋给局部变量 shouldSet。
· 将 Indices Box 的每一行拷贝到一个临时字符串, 并把这个字符串转换成一个整数, 把这个整数值存到索引数组的一个成员中。
· 使用一个循环把 Indices Box 控制的每一行读出, 并把该行转换成一个索引, 然后用这个索引来选中(或取消选中, 这取决于局域变量 shouldSet 的值) 一个列表框项。

小　　结

今天的课程介绍了列表框控制, 它允许用户从一个值的固定集合中选择。我们学习了以下内容:

· 单项列表框控制提供了供选择的项目列表。这种类型的列表框只允许每次选一项。
· 多选列表框允许在一个列表框中选中多项, 以便集中处理。在建立列表框时设定 LBS - EXTENDEDSEL 风格, 这样可以通过保持 Shift 键按下和单击鼠标来快速

地扩展选中项的范围。

问 与 答

问: 函数 CListBox::Dir 的 lpWildCard 参数的取值中, 可以包含多个通配符如 *.CPP *.H 吗?

答: 不可以。lpWildCard 的取值只限一个文件名通配符。

问: MFC 库支持对有关列表框控制中滚动消息的截获吗?

答: 不支持。

问: 如何使一个程序的两个列表框同步滚动?

答: 常用的方法为: 建立两个带隐藏滚动条的列表框, 然后加入一个滚动条控制(参阅第 19 天)。可以处理该控制发出的消息, 并相应地滚动列表, 这样就可以使列表框与滚动条控制同步。不能截获滚动条发出的消息, 滚动条是列表框的一部分。

问: 可以在有 LBS_SORT 风格设置的列表框中使用函数 InsertString 吗?

答: 不可以。因为 LBS_SORT 风格按顺序维护列表框项, 使用 InsertString 会破坏列表框的顺序, 应换以 AddString 成员函数。

作 业

作业部分所提供的测验题有助于巩固理解所学内容, 通过练习可以积累运用所学知识的经验。要尽量弄懂测验和练习的答案后再继续下一天的课程。附录 B 的"答案"中提供了解答。

测 验

1. 对还是错?
 列表框的通知消息 LBN_SETFOCUS 适用于有关选中一个列表框控制的任选的初始化过程。
2. 对还是错?
 列表框的通知消息 LBN_KILLFOCU 适用于对取消选中一个列表框控制后的任选的确认过程。
3. 对还是错?
 列表框只允许用 LBN_KILLFOCUS 通知检测最终选择。
4. 对还是错?
 应使用带特定标志的 LBN_CHANGE 去检测鼠标在一个列表框项上的双击事件。
5. 对还是错?
 LBS_STANDED 建立一个表项无序的列表框控制。

练 习

写一个程序显示一个与出现在文本框中的文件通配符相匹配的文件的列表(在列表框中)。

第19天

滚动条和组

第18天中介绍了列表框控制。今天要介绍两个类似的控制：滚动条和组合框。滚动条允许从一个大的取值范围中快速地选择一个数值；组合框控制含有编辑控制和列表框，允许用户从列表框元素中选中一个值，或者在编辑控制部分输入一个新值。我们将学习到以下的课题：

- 滚动条控制
- 各种风格的组合框控制

19.1 滚动条控制

Windows 允许滚动条与窗口、列表以及组合框结合在一起，也可以作为一个单独的控制而存在。滚动条控制的外观和行为都和窗口的滚动条很相像。控制有一个跟踪当前值的游标，允许鼠标按行或页移动它。游标的这种机制由 OnVScroll 或 OnHScroll 成员函数支持。另外，滚动条也响应光标控制键，如 Home, End, PageUp 和 PageDown，这种特性由成员函数 OnKeyDown 支持。滚动条的主要意图在于允许快速而高效地从一个预定的取值范围中选出一个整数值，例如，Windows 使用滚动条调整调色板、键盘响应速率和鼠标灵敏度。

CScrollBar 类

Microsoft 基本类(MFC)提供了 CWnd 的子孙类 CScrollBar，它是滚动条控制的模型类。CScrollBar 类定义了一个类构造函数和一些用于设置和查询控制当前位置和范围的成员函数，象其他 MFC 类的一样，CScrollBar 类的构造函数没有参数，建立一个滚动条控制时，应首先调用构造函数，随后调用 Create 成员函数。

语　法

函数 Create

函数 Create 声明如下：

```
BOOL Create(DWORD dwStyle,        // control style
            const RECT& rect,     // location and dimensions
            CWnd pParentWnd,      // pointer to parent window
            UINT nID);            // control ID
```

参数 dwStyle 指定滚动条控制的风格，滚动条可含有 WS_xxxx 窗口风格和特殊的 SBS_xxxx 滚动条风格，表 19.1 中列出了 SBS_xxxx 风格。你可以指定 SBS_SIZEBOX

风格，从而使 CScrolBar 实例成为大小可变的框，参数 rect 指定控制的位置和大小，参数 pParentWnd 为指向父窗口的指针，参数 nID 为滚动条实例的 ID。

示　例:

```
CScrollBar * pSB;
pSB- > Create(SBS - HORZ, rectDefault, this, ID - SB - SCR);
```

表 19.1　滚动条控制的 SBS - xxxx 风格

值	含　义
SBS - BOTTOMALIGN	该风格与 SBS - HORZ 风格一起使用，可使滚动条的底边和函数 Create 中确定的矩形底边对齐。
SBS - HORZ	指定水平滚动条，如果没有指定 SBS - BOTTOMALIGN 或 SBS - TOPALIGN 风格，则滚动条的位置、宽度和高度由函数 Create 的参数 rect 确定。
SBS - LEFTALIGN	该风格与 SBS - VERT 风格一起使用，使滚动条的左边和函数 Create 中确定的矩形框的左边对齐。
SBS - RIGHTALIGN	该风格与 SBS - VERT 风格一起使用，使滚动条右边与函数 Create 确定的矩形框的右边对齐。
SBS - SIZEBOX	指定一个改变大小的框，如果没有指定接下来的两个 SBS - xxxx 风格中的一个，则框的位置、宽度和高度由函数 Create 的参数 rect 确定。
SBS - SIZEBOXBOTTOMRIGHTALIGN	该风格与 SBS - SIZEBOX 风格一起使用，使尺寸框的右下角和函数 Create 中确定的矩形框的右下角对齐。
SBS - SIZEBOXTOPLEFTALIGN	该风格与 SBS - SIZEBOX 风格一起使用，使尺寸框的左上角和函数 Create 确定的矩形框的左上角对齐。
SBS - TOPALIGN	该风格 SBS - HORZ 风格一起使用，使滚动条的上边和函数 Create 中确定的矩形框的上边对齐。
SBS - VERT	指定垂直滚动条，若没有指定 SBS - RIGHTALIGN 或 SBS - LEFTALIGN 风格，则滚动条的位置、宽度和高度由函数 Create 的参数 rect 确定。

CScrollBar 声明了若干成员函数，主要有以下这些:

- 当建立一个 CScrollBar 实例后最有可能用到的第一个成员函数为 SetScrollRange，该函数允许给滚动条设置一个范围。
- 成员函数 GetScrollRange 允许查询滚动条的当前滚动范围。
- 无参数成员函数 GetScrollPos 返回游标的当前位置。
- 成员函数 SetScrollPos 移动游标到指定位置，但要保证新的标志位置在当前滚动条范围中。

语　法

函数 SetScrollRange

函数 SetScrollRange 声明如下:

```
void SetScrollRange(int nMinPos, int nMaxPos,
```

```
BOOL bRedraw= TRUE);
```

参数 nMinPos 和 nMaxPos 的值为滚动条控制的新的取值范围，布尔型参数 bRedraw 表示是否用新的范围重绘滚动条。

示　例：

```
int nMin= 1, nMax= 100;
CScrollBar*  pSB;
//create control...
B- > SetScrollRange (nMin, nMax);
```

语　法

函数 GetScrollRange

函数 GetScrollRange 声明如下：

```
void GetScrollRange(LPINT lPMinPos, LPINT lpMaxPos) const;
```

参数 lpMinPos 和 lpMaxPos 指向滚动条控制的当前范围。

示　例：

```
int nMin, nMax
CScrollBar*  pSB;
//create control ...
pSB- > SetScrollRange(&nMin, &nMax);
```

语　法

函数 SetScrollPos

成员函数 SetScrollPos 声明如下：

```
void SetScrollPos (int nPos, BOOL bRedraw= TRUE);
```

参数 nPos 指定新的游标位置，布尔型参数 bRedraw 表明是否用新的位置重绘滚动条实例。

示　例：

```
CScrollBar * pSB;
//create control ...
pSB- > SetScrollPos(nMedianValue);
```

响应滚动条通知消息

CScrollBar 滚动条控制实例依靠 CWnd::OnVScroll 或 CWnd::OnHScroll 成员函数支持游标的实际滚动，MFC 库要求把这些滚动函数作为父窗口的成员函数来声明。怎样使父窗口和滚动条联系起来呢？答案在于 OnVScroll 和 OnHScroll 成员函数的参数。

```
afx_msg void OnVScroll(UINT nSBCode, UINT nPos,
CScrollBar*  pScrollBar);
afx_msg void OnHScroll(UINT nSBCode, UINT nPos,
CScrollBar*  pScrollBar);
```

参数 pScrollBar 为指向用户击中的滚动条控制的指针。在第 15 天课程中的滚动窗口例子中，我们使用了 OnVScroll 和 OnHScroll 函数，但在函数的语句中没有参数

pScrollBar。当编写用于处理一个滚动条控制的 OnVScroll 或 OnHScroll 成员函数时，就必须和其他 CScrollBar 成员函数一起使用 pScrollBar 参数，在运行中消息处理系统应使滚动条控制和父窗口的 OnVScroll 或 OnHScroll 成员函数联系在一起，接下来的例子将演示这种方式如何工作。

倒计数计时器

倒计数计时器应用程序用到如下的控制：

- 一个用于接收计时器的输入和显示当前计时器值的 Timer Input Box 编辑控制。
- 用于标名编辑框的静态文本控制。
- 用于触发倒计数计时器的 Start 按钮。
- Exit 按钮。
- 一个计时器滚动条控制，缺省范围为 0 到 600 秒。
- 用于标名计时器滚动条的取值范围的静态文本控制。

可以用两种设定秒数的方式中的任一种：在编辑控制中键入秒数或使用滚动条。当移动滚动条游标时，当前的游标位置就会出现在编辑框中，触发计数过程，可击中 Start 按钮或按 Alt-S 键。倒计数过程使用存储在编辑框中的值，并把它转换为计数的最大的秒数。如果编辑框中内容为空、0 或其他非数值文本，程序将赋给一个缺省值，即 15 秒。在计数期间，应用程序减少编辑框中的秒数，向上移动滚动条游标，当倒计数终止时，程序发出嘟声，恢复滚动条上限值，并恢复上限静态文本。

倒计数计时器应用程序演示了如下的滚动条操作：

- 设置和改变滚动条的取值范围。
- 移动和改变滚动条游标位置(程序演示了怎样内部执行这些任务或使用鼠标)。
- 利用滚动条提供一个值。

清单 19.1 列出了头文件 CTLSCRL1.H 的源代码，清单 19.2 为资源描述文件 CTRLSCRL1.RC，它含有加速键表和菜单资源，清单 19.3 中列出了程序文件 CTRLSCRL1.CPP 的源代码，程序使用一个带单菜单项 Exit 的菜单。

键 入 **清单 19.1 头文件 CTRLSCRL1.H 的源代码**

```
1: # define CM_EXIT (WM_USER + 100)
2: # define ID_START_BTN (WM_USER + 101)
3: # define ID_EXIT_BIN (WM_USER + 102)
4: # define ID_INPUT_BOX (WM_USER + 103)
5: # define ID_TIME_SCR (WM_USER + 104)
```

键 入 **清单 19.2 资源描述文件 CTRLSCRL1.RC**

```
1: # include< windows.h>
2: # include < afxres.h>
```

```
# include "ctlscrl1.h"

BUTTONS ACCELERATIORS
BEGIN
"s", ID_START_BTN, ALT
"e", ID_EXIT_BTN, ALT
END

EXITMENU MENU LOADONCALL MOVEABLE PURE DISCARDABLE
BEGIN
MENUITEM "E&xit", CM_EXIT
END
```

键　入　　清单 19.3　程序文件 CTRLSCRL1.CPP 的源代码

```
# include <stdio.h>
# include <string.h>
# include <afxwin.h>
# include "ctlscrl1.h"

//current timer limit is 10 minutes
const MaxTimer = 600; // seconds
const MaxEditLen= 10;
const PageSize= 100;

class CxButton:public CButton
{
public:

   BOOL Create(const char FAR * lpCaptions, const RECT& rect,
               CWnd* pParentWnd, UINT nID)
   {
      return CButton::Create(lpCaption,
                  WS_CHILD|WS_VISIBLE|BS_PUSHBUTTON,
                  rect, pParentWnd, nID);
   }
};

class CWindowApp: public CWinApp
{
public:
     virtual BOOL InitInstance();
};

// expand the functionality of CFrameWnd by deriving
// class CAppWindow
class CAppWindow: public CFrameWnd
{
public:

```

```
36:    CAppWindow();
37:    ~CAppWindw();
38:
39: protected:
40:    // declare the pointers to the various controls
41:    CEdit*  InputBox;
42:    CxButton*  StartBtn;
43:    CxButton*  ExitBtn;
44:    CScrollBar*  TimerScr;
45:    CStatic*  InputTxt;
46:    CStatic*  Timer1Txt;
47:    CStatic*  Timer2Txt;
48:    // handle creating the controls
49:    afx_msg int OnCreate(LPCREATESTRUCT lpCS);
50:    // handle the vertical scrolling of the scroll bar
51:    afx_msg void OnVScroll(UINT nSBCode, UINT nPos,
52:                    CScrollBar*  pScrollBar);
53:    // handle starting the timer
54:    afx_msg void HandleStartBtn();
55:    //handle starting the timer
56:    afx_msg void CMStartBtn()
57:       { HandleStartBtn(); }
58:    //handle exiting the program
59:    afx_msg void HandleExitBtn()
60:       { SendMessage(WM_CLOSE); }
61:    // handle exiting the program
62:    afx_msg void CMExitBtn()
63:       { SendMessgae (WM_CLOSE); }
64:    // handle moving the scroll bar
65:    void HandleTimerScr();
66:    // handle exiting the application
67:    afx_msg void OnExit()
68:        { SendMessage(WM_CLOSE); }
69:    // handle closing the window
70:    afx_msg void OnClose();
71:    void makeRect(int nX,int nY, int nW, int nH,CRect&r);
72:
73: private:
74: // delay the program for about ms milliseconds
75:    void delay(DWORD dwMs);
76:
77: // declare message map macro
78:    DECLARE_MESSAGE_MAP();
79: };
80:
81: CAppWindow::CAppWindow()
82: {
83:    //   load accelerator resources
84:    LoadAccelTable("BUTTONS");
85:    // create the window
86:    Create(NULL,"CountDown Timer", WS_OVERLAPPEDWINDOW,
87:        rectDefault, NULL, "EXITMENU");
88 }
89:
```

```
90: CAppWindow::~CAppWindow()
91: {
92:    delete InputBox;
93:    delete StartBtn;
94:    delete ExitBtn;
95:    delete TimerScr;
96:    delete InputTxt;
97:    delete Timer1Txt;
98:    delete Timer2Txt;
99: }
100:
101: int CAppWindow::OnCreate(LPCREATESTRUCT lpCS)
102: {
103:    char s[81];
104:    int nX= 50,nY= 50;
105:    DWORD dwStaticStyle =  WS_CHILD©WS_VISIBLE©SS_LEFT;
106:    DWORD dwEditStyle =  WS_CHILD©WS_VISIBLE©ES_LEFT©
107:                         WS_BORDER;
108:    DWORD dwScrollStyle =  WS_CHILD©WS_VISIBLE©SBS_VERT;
109:    CRect r;
110:
111:    // create the timer input box and its label
112:    makeRect(nX,nY,150, 30,r);
113:    InputTxt= new CStatic();
114:    InputTxt->Create("Timer Input Box",dwStaticStyle,r,this);
115:    nY += 30 + 5;
116:    makeRect(nX,nY,150, 30,r);
117:    InputBox= new CEdit();
118:    InputBox->Create(dwEditStyle,r,this,ID_INPUT_BOX);
119:    InputBox->LimitText(MaxEditLen);
120:    // create the Start button
121:    nY += 30 + 20;
122:    makeRect(nX,nY,60, 40,r);
123:    StartBtn= new CxButton();
124:    StartBtn->Create("&Start",r,this,ID_START_BTN);
125:    // create the Exit button
126:    nX += 60+ 20;
127:    // makeRect(nX,nY,60, 40,r);
128:    ExitBtn= new CxButton();
129:    ExitBtn->Create("&Exit",r,this,ID_EXIT_BTN);
130:    //create the timer scroll bar
131:    nX= 300;
132:    nY= 100;
133:    makeRect(nX,nY,20, 150,r);
134:    TimerScr= new CScrollBar();
135:    TimerScr->Create(dwScrollStyle,r,this,ID_TIMER_SCR);
136:    // create the static text controls that label the
137:    //minimum and maximum values
138:    nX += 20 + 10;
139:    makeRect(nX,nY,80, 20,r);
140:    Timer1Txt= new CStatic();
141:    Timer1Txt->Create("0",dwStaticStyle,r,this);
142:    nY += 130;
143:    makeRect(nX,nY,80, 20,r);
```

```
144:    sprintf(s, "%d", MaxTimer);
145:    Timer2Txt= new CStatic();
146:    Timer2Txt- > Create(s, dwStaticStyle, r, this);
147:    //initialize the timer range
148:    TimerScr- > SetScrollRange(0, MaxTimer);
149:    return CFrameWnd::OnCreate(lpCS);
150: }
151:
152: void CAppWindow::HandleStartBtn()
153: {
154:    char s[MaxEditLen+ 1];
155:    int nX;
156:
157:    // get the text in the edit box
158:    InputBox- > GetWindowText(s, MaxEditLen);
159:    // convert the string into an integer
160:    nX = atoi(s);
161:    // if nX is 0 assign it 15
162:    nX = (nX ! = 0) ? nX: 15;
163:    // set the maximum timer static text
164:    sprintf(s, "%d", nX);
165:    Timer2Txt- > SetWindowText(s);
166:    // set the new range
167:    TimerScr- > SetScrollRange(0, nX);
168:    // set the thumb position to the maximum position
169:    TimerScr- > SetScrollPos(nX);
170:    // countdown loop
171:    while (nX- - > 0) {
172:       delay(980);
173:       // update the thumb position
174:       TimerScr- > SetScrollPos(TimerScr- > GetScrollPos()- 1);
175:       sprintf(s, "%d", nX);
176:       // echo thumb position in the edit box
177:      InputBox- > SetWindowText(s);
178:      InputBox- > UpdateWindow();
179:   }
180:   MessageBeep(0);//beep
181:   // restore the default timer limits
182:   sprintf(s, "%d", MaxTimer);
183:   TimerScr- > SetScrollRange(0, MaxTimer);
184:   TimerScr- > SetScrollRos(MaxTimer);
185:   Timer2Txt- > SetWindowText(s);
186:   }
187:
188: void CAppWindow::OnVScroll(UINT nSBCode, UINT nPos,
189:                                  CScrollBar* pScrollBar)
190: {
191:    short nScrollInc;
192:    int nNewPos;
193:    int nVScrollMin, nVScrollMax;
194:    int nVScrollPos= pScrollBar- > GetScrollPos();
195:
196:    // get the scroll bar range
197:    pScrollBar- > GetScrollRange(&nVScrollMin, &nVScrollMax);
```

```
198:    switch(nSBCode) {
199:      case SB_TOP://move to the top
200:           nScrollInc= - nVScrollPos;
201:           break;
202:      case SB_BOTTOM://move to the bottom
203:           nScrollInc= nVScrollMax- nVScrollPos;
204:           break;
205:      case SB_LINEUP://move one line up
206:           nScrollInc= - 1;
207:           break;
208:      case SB_LINEDOWN://move one line down
209:           nScrollInc= 1;
210:           break;
211:      case SB_PAGEUP://move one page up
212:           nScrollInc= - PageSize;
213:           break;
214:      case SB_PAGEDOWN://move one page down
215:           nScrollInc= PageSize;
216:           break;
217:      case SB_THUMBPOSITION://track the thumb box
218:           nScrollInc= nPos- nVScrollPos;
219:           break;
220:      default:
221:           nScrollInc= 0;
222:    }
223:    // calculate new vertical thumb position so that:
224:    // 0< = nNewPos< = nVScrollMax
225:    nNewPos= max(0, min(nVScrollPos+ nScrollInc, nVScrollMax));
226:    //adjust scroll increment
227:    nScrollInc= nNewPos- nVScrollPos;
228:    //is nScrollInc not zero?
229:    if (nScrollInc) {
230:      // move the thumb box
231:      pScrollBar- > SetScrollPos(nNewPos);
232:      //scroll the list boxes
233:      HandleTimerScr();
234:    }
235: }
236:
237: void CAppWindow::HandleTimerScr()
238: {
239:    int nX= TimerScr- > GetScrollPos();
240:    char s[MaxEditLen+ 1];
241:    // convert the thumb position into a string
242:    sprintf(s,"%d",nX);
243:    // insert the string in the edit box
244:    InputBox- > SetWindowText(s);
245: }
246:
247: void CAppWindow::OnClose()
248: {
249:    if (MessageBox("Want to close this application",
250:                   "Query",MB_YESNO ©¦MB_ICONQUESTION)= = IDYES)
251:      DestronyWindow();
```

```
252: }
253:
254: void CAppWindow::delay(DWORD dwMs)
255: {
256:    DWORD dwTime= GetTickCount();
257:    do { ; } while ((GetTickCount() -  dwTime) <  dwMs);
258: }
259:
260: void CAppWindow::makeRect(int nX, int nY, int nW, int nH,
261:                           CRect& r)
262: {
263:    r.top =  nY;
264:    r.left =  nX;
265:    r.bottom =  nY +  nH;
266:    r.right =  nX +  nW;
267: }
268:
269: BEGIN_MESSAGE_MAP(CAppWindow, CFrameWnd)
270:    ON_WM_CREATE()
271:    ON_WM_VSCROLL()
272:    ON_BN_CLICKED(ID_START_BTN, HandleStartBtn)
273:    ON_COMMAND(ID_START_BTN, CMStartBtn)
274:    ON_BN_CLICKED(ID_EXIT_BTN, HandleExitBtn)
275:    ON_COMMAND(ID_EXIT_BTN, CMExitBtn)
276:    ON_COMMAND (CM_EXIT, OnExit)
277:    ON_WM_CLOSE()
278: END_MESSAGE_MAP()
279:
280: // Construct the CWindowApp's m_pMainWnd data member
281: BOOL CWindowApp::InitInstance()
282: {
283:    m_pMainWnd= new CAppWindow();
284:    m_pMainWnd->ShowWindow(m_nCmdShow);
285:    m_pMainWnd->UpdateWindow();
286: return TRUE;
287: }
288:
289: // application's constructor initializes and runs the app
290: CWindowApp WindowApp;
```

图 19.1 为应用程序 CTLSCR1.EXE 的一个运行范例。

分　析

清单 19.3 中的应用程序中声明了应用程序类 CWindowApp、窗口类 CAppWindow 和按钮控制类 CxButton。窗口类 CAppWindow 中声明了若干指向应用程序控制的指针作为数据成员，类 CAppWindow 声明了一个构造函数、一个析构函数和 10 个成员函数。

CAppWindow 类构造函数建立应用程序窗口，并执行如下的任务：

- 加载加速键资源。
- 建立应用程序窗口，指定标题、缺省的尺寸、缺省的位置和 EXITMENU 菜单资

图 19.1　应用程序 CTLSCR1.EXE 的一个运行范例

源。

第 101 到 150 行定义了成员函数 OnCreate，它建立应用程序使用的各种控制，函数执行以下的任务：

- 建立计时器编辑框和及标名它的的静态文本。
- 建立标名为 Start 和 Exit 的下压按钮控制。
- 建立计时器滚动条控制（函数 CScrollBar::Create 指定一个带 SBS_VERT 风格的竖直滚动条，并把 ID_TIMER_SCR 作为控制的 ID）。
- 建立标名滚动条范围的静态文本控制，最小值总为 0，最大值用全局常量 MaxTimer 进行初始设置（程序改变倒计数过程的上限）。
- 返回本函数的结果。

第 152 到 186 行定义了成员函数 HandleStartBtn，用于提供触发计时器倒计数的功能，该函数执行如下的任务：

- 检索编辑框中的文本，并把它存到局部变量 s 中。
- 把变量 s 中的字符转换成一个整型值并存储到局部变量 x 中。
- 检查 x 的值（如为 0，此函数给变量 x 赋值为 15）。
- 设置计时器上限静态文本为变量 x 中的值所对应的字符串映像。
- 调用以 0 和 x 参数值的 CScrollBar::SetScrollRange 成员函数，设置计时器滚动条的范围。
- 调用成员函数 CScrollBar::SetScrollPos 将游标移动到滚动条的底部（函数调用时以 x 为变元）。
- 开始倒计数循环，循环用到了延时函数，让程序等待 980 毫秒，随后游标位置向上

移动一个单位值(使用成员函数 CScrollBar ::GetScrllPos 和 CScrollBar ::SetScrollPos),并把当前游标位置的字符串映像写进编辑框中(这样可模拟编辑框按秒显示倒计数的时间)。

· 循环中止时发出嘟声。

· 恢复滚动条计时器的缺省上限值 600,并据此更新上限静态文本。

成员函数 CMStartBtn 捕获 Alt- S 键产生的命令消息,函数仅调用 HandleStartBtn。

成员函数 HandleExitBtn 和 CMExitBtn 成员函数发送一条 WM - CLOSE 消息给应用程序窗口。

注　意　第 188 到 235 行定义了成员函数 OnVScroll,负责垂直滚动条控制的游标。函数使用 pScrollBar 函数查询和设置游标位置,也查询当前滚动条范围。

第 237 到 245 行定义了成员函数 HandleTimerScr,该函数被 OnVscroll 成员函数调用。函数把游标的位置转换成字符串,并写入编辑框中。这样,当移动游标时,函数 HandleTimerScr 负责更新编辑框中的内容。

延时成员函数调用 API 函数 GetTickCountAPT 产生模拟所需要的时延。

19.2　组合框控制

Windows 支持将编辑框和列表框组合在一起的组合框控制。这样,组合框允许在列表框元素中选中一个项,还可键入自选的内容。从某种意义上说,组合框的列表框部分含有使用较方便的或经常使用的选项。组合框不像列表框,它不会限制仅选择列表框中的项。组合框有三种类型:简单型、下拉型和下拉列表型。

新　词

简单组合框(simple combo box)

简单组合框中的编辑框和列表框总是可见的。下拉组合框不同于简单类型之处在于,只有当击中下滚箭头时,列表框才出现,下拉列表组合框也是当击中下滚箭头时,出现一个下拉列表,但这种类型的组合框里没有编辑框。

CComboBox 类

MFC 提供了 CComboxBox 类,它是 CWnd 的派生类,支持组合控制。CComboxBox 类声明了一个构造函数和众多的成员函数,用于支持列表框和编辑控制组元。Create 成员函数协助类构造函数建立类实例

语　法

函数 Create

函数 Create 声明如下:

```
BOOL Create( DWORD dwStyle        // control style
             const RECT& rect,    // location and dimensions
```

```
CWnd*  pParentWnd,  // pointer to parent window
UINT nID);          // control ID
```

参数 dwStyle 指定列表框控制的风格,列表框风格包括通常的 WS_xxxx 窗口风格以及专门的 CBS_xxxx 组合框风格(见表 19.2);参数 rect 指定控制的位置和大小;参数 pParentWnd 为指向父窗口的指针;参数 nID 为滚动条实例的 ID。

示　例:

```
DWORD dwStyle = WS_CHILD|WS_VISIBLE|WS_GROUP|WS_TABSTOP
                |CBS_SORT|CBS_AUTOHSCROLL|WS_VSCROLL;
CComboBox pCB;
pCB->Create(dwStyle, rectDefault, this, ID_CTL_CMB);
```

参数 dwStyle 的取值可以包括以下三种风格中的一种:用于简单组合框的 CBS_SIMPLE、用于下拉组合框的 CBS_DROPDOWN 和用于下拉列表组合框的 CBS_DROPDOWNLIST。一个典型的 CComboBox 实例带有风格 WS_CHILD, WS_VISIBLE, WS_GROUP, WS_TABSTOP, CBS_SORT, CBS_AUTOHSCROLL 和 WS_VSCROLL。

表 19.2　组合框控制风格

风　格	含　义
CBS_AUTOHSCROLL	当在行尾输入一个字符时,将编辑控制中的文本自动向右滚动(若去掉本风格,文本的字符只能限制在编辑控制的矩形边界内)。
CBS_DROPDOWN	指定一个下拉组合框。
CBS_DROPDOWNLIST	指定一个下拉列表组合框。
CBS_SIMPLE	指定一个简单组合框。
CBS_SORT	自动排序列表框中的项。

CComboBox 类声明了成员函数,用于管理列表框和编辑控制组元。这些函数中的大多数与 CEdit 和 CListBox 类的成员相似。因为 MFC 库没有把 CComboBox 声明为 CListBox 和 CEdit 的子孙类(使用多重继承),因而这些函数必须重新声明:

其中,处理列表框组元的函数如下:

- 无参数成员函数 GetCount 返回列表框中的项数。
- 无参数成员函数 GetCurSel 返回当前选项的索引。如没有当前选项,函数返回 CB_ERR。
- 成员函数 SetCurSel 设置当前选中项。
- 重载成员函数 GetLBText 与函数 CListBox::GetText 相似,本函数返回列表框的表项。
- 成员函数 GetLBTextLen 返回指定索引的列表框表项的长度。
- 成员函数 ShowDropDown 显示或隐藏一个组合框控制的列表框部分。
- 成员函数 AddString 在组合框控制的列表框中增加一个字符串。
- 成员函数 DeleteString 从列表框中删除一项。

· 成员函数 InsertString 把一字符串插入列表框中指定的位置。
· 无参数成员函数 ResetContent 清除组合框控制的列表框。
· 成员函数 Dir 是一个特殊的成员函数,用于在列表框中插入文件名。
· 成员函数 FindString 在列表框中大小写敏感地搜寻一个项。
· 成员函数 SelectString 选择一个与搜索字符串匹配的列表框项。

语　法

函数 SetCurSel

函数 SetCurSel 声明如下:

```
int SetCurSel(int nSelect);
```

参数 nSelect 指定列表框中新的选中项的索引。本函数返回选中项中的索引,若出错返回 CB - ERR。

示　例:

```
CComboBox * pCB;
//create control ....
pCB- > SetCurSel(0); // select the first item
```

语　法

函数 GetLBText

函数 GetLBText 声明如下:

```
int GetLBText(int nIndex, char FAR* lpItem) const;
void GetLBText(int nIndex, CString& rString) const;
```

参数 nIndex 指定检索项的索引,lpItem 指向接收表项的缓冲区(必须足够大),rString 是接收表项拷贝的 CString 对象的引用。此函数的第一种形式返回检索到的表项长度,在出错时返回 CB - ERR。

示　例:

```
char * s[81];
CComboBox * pCB;
// create control...
pCB- > GetLBText(0, s); // get string in first list box item
```

语　法

函数 GetLBTextLen

函数 GetLBTextLen 声明如下:

```
int GetLBTextLen(int nIndex) const;
```

参数 nIndex 指定列表框项,函数返回查找到的列表框项的长度。如果 nIndex 的取值无效,返出 CB - ERR。使用 GetLBTextLen 可确保函数 GetLBText 使用的缓存区足够大。

示　例:

```
int nLen;
CComboBox* pCB;
// create control ...
nLen = pCB- > GetLBTextLen(0);  //get length of first item
```

语　法

函数 ShowDropDown

函数 ShowDropDown 声明如下:

```
void showDropDown(BOOL bShowIt= TRUE);
```

布尔型参数 bShowIt 是一个用于显示(为 TRUE 时)或隐藏(为 FALSE 时)下拉列表框的开关。当想在一个程序中切换下拉列表框的可见性时,使用 ShowDropDown 成员函数。

示　例:

```
CComboBox * pCB;
// create control....
pCB- > ShowDropDwon();
```

语　法

函数 AddString

函数 AddString 声明如下:

```
int AddString(const char FAR * lpString);
```

参数 lpString 为指向加入字符串的指针。函数成功时,返回插入项的索引,否则,函数返回负的 CB - ERR 值。如果设置了 CBS - SORT 风格,增加字符串放进排序表中适当位置,否则,字符串加入表尾。

示　例:

```
CComboBox* pCB;
// create control....
pCB- > AddString("MS- DOS");
```

语　法

函数 DeleteString

函数 DeleteString 声明如下:

```
int DeleteString(int nIndex);
```

参数 nIndex 为要删除的表项的索引,如果删除成功,函数返回剩余字符串数;否则,本函数返回一个负的 CB - ERR 值。

示　例:

```
CComboBox* pCB;
//create control...
//add strings to control...
pCB- > DeleteString(0);  //delete first item
```

语　法

函数 InsertString

函数 InsertString 声明如下:

```
int InsertString(int nIndex, const FAR* lpString);
```

参数 nIndex 为插入索引,参数 lpString 为指向插入字符串的指针,当设定 CBS - SORT 风格的时候不要使用 InsertString,除非插入的项不会破坏列表项的次序。本函数返回插入字符串的索引,出错时返回 CB - ERR,如果列表框用完了内存空间,函数返回 CB - ERRSPACE,

示　例:

```
CComboBox * pCB;
//create control...
//add strings to control...
pCB- > InsertString( 0, "Windows");  //insert as new first item
```

语　法

函数 Dir

函数 Dir 声明如下:

```
int Dir(UINT attr, const char FAR* lpWildCard);
```

参数 attr 为属性的组合,文件的属性和 CListBox::Dir 函数中的一样,在第 18 天的表 18.2 中已列出。参数 lpWildCard 指向一个文件名通配符,如*.*,L*.EXE 或 A???.CPP 等

示　例:

```
int nNumFiles;
CComboBox * pCB;
// create control...
nNumFiles = pCB- > Dir(-A-ARCH, "*.*");
```

语　法

函数 FindString

函数 FindString 的声明如下:

```
int FindString(int nStartAfter, const char FAR* lpItem) const;
```

参数 nStartAfter 为要搜索的第一个列表项成员的索引,参数 lpItem 为指向要搜索的字符串的指针。函数从位置 nStartAfter 开始搜寻整个列表,如果需要还可从表头继续搜索。当发现一个列表成员与搜索字符串匹配或全部列表都搜索过,则搜索停止。若 nStartAfter 取值为-1,则强制函数从表头开始搜索,函数返回匹配表项的位置,如未发现匹配项或出错则返回 LB-ERR

示　例:

```
int nIndex
CComboBox * pCB;
//create control...
//add strings to control...
Index = pCB- > FindString(-1, "MS-DOS");
```

语　法

函数 SelectString

函数 SelectString 声明如下:

```
int SelectString (int nStartAfter, const char FAR* lpItem);
```

SelectString 的参数和搜索机制与 FindString 的一样。不同的是,SelectString 选中与参数 lpItem 存取的字符串相匹配的表项

示　例:

```
CComboBox * pCB;
//create control...
```

```
//add strings to contrl...
pCB->SelectString(-1,"MS-DOS");
```

下面的函数管理编辑控制部分:

- 无参数成员函数 GetEditSel 返回一个含有被选中文本起始和终止字符位置的 DWORD 值。低位字为起始字符位置(第一个选中字符的索引),高位字为终止字符位置(紧接在选中文本后的第一个未选中字符的索引)。
- 成员函数 LimitText 限制在组合框控制的编辑框中可以输入的字符数。
- 无参数成员函数 Clear 清除选中的文本。
- 无参数成员函数 Copy 把选中的文本复制到剪贴板中。
- 成员函数 Cut 清除选中文本,并把它送到剪贴板中。
- 成员函数 Paste 把剪贴板中的内容复制到编辑框的当前插入点。

语　法

函数 LimitText

函数 LimitText 声明如下:

```
BOOL LimitText(int nMaxChars);
```

参数 nMaxChars 指定编辑框中文本的界限。如成功,本函数返回 TRUE,否则返回 FALSE。

示　例:

```
const int MAX_STR_SIZE = 81
CComboBox* pCB;
// create control...
// add strings to control...
pCB->LimitText(MAX_STR_SIZE);
```

响应组合框通知消息

组合框控制发送各种类型的消息,如表 19.3 中所示。这个表也列出了关联各种命令和通知消息的消息映射宏。每一种命令或通知消息都要求在控制的父窗口类单独声明一个成员函数。

表 19.3　组合框通知消息

消　息	宏	含　义
WS_COMMAND	ON_COMMAND	Window 命令消息。
CBN_SELCHANGE	ON_CBN_SELCHANGE	用鼠标单击选中一个组合项。
CBN_DBLCLK	ON_CBN_DBLCLK	用鼠标双击选中一个组合项。
CBN_SETFOCUS	ON_CBN_SETFOCUS	组合框获得输入焦点。
CBN_KILLFOCUS	ON_CBN_KILLFOCUS	组合框失去输入焦点。
CBN_ERRSPACE	ON_CBN_ERRSPACE	组合框不能分配足够内存容纳新的列表项。
CBN_EDITCHANGE	ON_CBN_EDITCHANGE	编辑框的内容已改变。
CBN_EDITUPDATE	ON_CBN_EDITUPDATE	编辑框的内容已更新。

组合框用作历史列表框

组合框可以是一个历史列表框。历史列表框通常遵循这些操作规则:

1. 组合框去掉 CBS - SORT 风格,按时间顺序插入列表项。新的项插入位置 0 处,把旧的项向下推。最老的项在表的底部。
2. 历史框通常会对可以插入的项的数目进行限制,以防止无节制使用内存。在这种方式下,当列表项的数目达到上限时,要删除最老的列表项。
3. 如果编辑控制中有一个串在对应列表框中不能找到与它严格匹配的字符串,那么编辑控制字符串作为一个新成员插入位置 0。
4. 如果编辑控制有一个串在对应列表框中能找到一个与它严格匹配的字符串,那么匹配的列表成员将移到位置 0,即表头。当然,这种处理先要从它当前位置删除匹配的的列表成员,然后把它插入位置 0。

新　词

历史列表框(history list box)
历史列表框是一种特殊的组合框。它按年代顺序在列表框中插入新的编辑控制字符串。

历史列表框是一种真正的组合框,它按一定方式操作其编辑控制和列表框项,不需要从 CComboxBox 派生出一个子孙类,以增加新的成员函数。

新　词

队列数据结构(queue data structure)
历史列表框的行为像一个队列数据结构,新项插入位置 0,将旧表项在列表中往下推。

COCA 应用程序第 4 版

下面看看计算器应用程序(COCA)的第 4 版。新的版本是从第 2 版而非第 3 版派生出来的(包含先前的结果替换操作数控制的# 字符的特性),图 19.2 为应用程序 CTLCOMB1.EXE 的运行范例。可以看出,其中使用了如下的控制:

- 下拉组合框 Operand1,其操作与历史列表框一样。
- 简单组合框 Operator,里面包含支持运算符的列表和函数 log, exp 和 sqrt(程序处理本控制发出的 LBN - KILLFOCUS 通知消息,选择这些函数只需分别敲入字母 l, e 和 s)。
- 下拉组合框 Operand2,其操作与历史列表框一样。
- 下拉组合框 Result,其操作与历史列表框一样。
- Error Message 编辑框。
- Calc 下压按钮控制。
- Exit 下压按钮控制。

图 19.2 为应用程序 CTLCOMB1.EXE 的一个运行范例。

编译和运行本程序,输入数值和运算符或函数进行运算。请注意,组合框的操作数和结果按时间顺序进入其相应的列表框中。操作域组合框可记忆输入的 30 个不同的操作数,Result 组合框记忆 30 个不同的计算结果,在某种意义上,Result 组合框充当临时过渡

图 19.2　应用程序 CTRLCOMB1.EXE 的一个运行范例

内存。

清单 19.4 中列出头文件 CTLCOMB1.H 的源代码，清单 19.5 列出资源描述文件 CTLCOMB1.RC。清单 19.6 包含程序文件 CTLCOM1.CPP 的源代码。资源文件定义了加速键和菜单资源。

键　入　　清单 19.4　头文件 CTLCOMB1.H 的源代码

```
1: # define CM_EXIT (WM_USER + 100)
2: # define ID_CALC_BTN 110
3: # define ID_EXIT_BTN 111
```

键　入　　清单 19.5　资源描述文件 CTLCOMB1.RC

```
 1: # include < windows.h>
 2: # include < afxres.h>
 3: # include "ctlcomb1.h"
 4:
 5: BUTTONS ACCELERATORS
 6: BEGIN
 7: "s", ID_START_BTN, ALT
 8: "e", ID_EXIT_BTN, ALT
 9: END
10:
11: EXITEMENU MENU LOADONCALL MOVEABLE PURE DISCARDABLE
12: BEGIN
```

```
13: MENUITEM "E&xit", CM - EXIT
14: END
15:
```

键　入　　清单 19.6　程序文件 CTLCOMB1.CPP 的源代码

```
 1: # include < stdlib.h>
 2: # include < ctype.h>
 3: # include < stdio.h>
 4: # include < math.h>
 5: # include < string.h>
 6: # include < afxwin.h>
 7: # include "ctlcomb1.h"
 8:
 9: //declare the constants that represent the sizes of the controls
10: const Wlb1= 100;
11: const Hlbl= 20;
12: const LblVertSpacing = 5;
13: const LblHorzSpacing = 40;
14: const Wbox= 100;
15: const Hbox= 30;
16: const BoxVertSpacing = 20;
17: const BoxHorzSpacing = 40;
18: const WLongbox= 4* (Wbox+ BoxHorzSpacing);
19: const Hbtn= 30;
20: const Wbtn= 80;
21: const BtnHorzSpacing= 30;
22: const BtnVertSpacing= 10;
23: const Wcmb= 100;
24: const Hcmb= 150;
25: const CmbVertSpacing= 10;
26: const CmbHorzSpacing= 40;
27: //declare maximum size of text in edit control
28: const MaxEditLen= 30;
29: //maximum number of items in a combo box that doubles up as
30: //history list box
31: const MaxHistory= 30;
32: //declare the ID_XXXX constants for the edit boxes
33: # define ID_OPERAND1_CMB 101
34: # define ID_OPERATOR_CMB 102
35: # define ID_OPERAND2_CMB 103
36: # define ID_RESULT_CMB 104
37: # define ID_ERRMSG_EDIT 105
38:
39: class CxButton:  public CButton
40: {
41: public
42:
43:    BOOL Create(const char FAR* lpCaption, const RECT& rect,
44:                CWnd* pParentWnd, UINT nID)
45:    {  return CButton::Create (lpCaption,
```

```
46:                     WS_CHILD ©¦WS_VISIBLE ©¦WS_TABSTOP,
47:                     rect, pParentWnd, nID);
48:    }
49: };
50:
51: // expand the functionality of CFrameWnd by deriving
52: // class CAppWindow
53: class CAppWindow：  public CFrameWnd
54: {
55: public:
56:
57:    CAppWindow();
58:    ～CAppWindow();
59:
60: protected;
61:    // declare the pointers to the controls
62:    CStatic*    Operand1Txt;
63:    CStatic*  OperatorTxt;
64:    CStatic*  Operand2Txt;
65:    CStatic*  ResultTxt;
66:    CStatic*  ErrMsgTxt;
67:    CComboBox*  Operand1Cmb;
68:    CComboBOx*  OperatorCmb;
69:    CComboBox*  Operand2Cmb;
70:    CComboBox*  ResultCmb;
71:    CEdit*  ErrMsgBox;
72:    CxButton*  CalcBtn;
73:    CxButton*  ExitBtn;
74:    // math error flag
75:    BOOL bInError;
76:
77:    //handle notification messages form the Operator combo box
78:    afx_msg void HandleOperatorCmb();
79:    //handle the calculation
80:    afx_msg void HandleCalcBtn();
81:    // handle the accelerator key for the Calculate button
82:    afx_msg void CMCalcBtn()
83:       { HandleCalcBtn(); }
84:    // handle exiting the application
85:    afx_msg void HandleExitBtn()
86:       { SendMessage (WM_CLOSE); }
87:    // handle the accelerator key for the Exit button
88:    afx_msg void CMExitBtn()
89:       { SendMessage(WM_CLOSE); }
90:    // handle exiting the application
91:    afx_msg void OnExit()
92:       { SendMessage (WM_CLOSE); }
93:    // handle creating the controls
94:    afx_msg int OnCreate(LPCREATESTRUCT lpCS);
95:    // initialize the instances of CAppWindow
96:    void InitAppWindow();
97:    // handle closing the window
98:    afx_msg void OnClose();
99:    // return a reference to memeber r based on individual
```

```
100:    // coordinates and dimenions
101:    void makeRect(int nX, int nY, int nW, int nH, CRect& r);
102:    // update the combo box with the text in the
103:    // accompanying edit box, assign that the text
104:    // is not already in the box
105:    void updateComboBox (CComboBox*  pComboBox);
106:
107:    // declare message map macro
108:    DECLARE - MESSAGE - MAP();
109: };
110:
111: class CWindowApp: public CWinApp
112: {
113: public:
114:    virtual BOOL InitInstance();
115: };
116: CAppWindow::CAppWindow()
117:
118: {
119:    // load accelerator resources
120:    LoadAccelTable("BUTTONS");
121:    // create the window
122:    Create (NULL,
123:       "Command-  Oriented Calculator Application(COCA) Version 3",
124:         WS - OVERLAPPEDWINDOW ©¦WS - MAXIMIZE,
125:            rectDefault, NULL, "EXITMENU");
126:    // clear the bInError flag
127:    bInError= FALSE;
128:    // initialize application
129:    InitAppWindow();
130: }
131:
132: CAppWindow::～CAppWindow()
133: {
134:    // delete the controls
135:    delete Operand1Txt;
136:    delete OperatorTxt;
137:    delete Operand2Txt;
138:    delete ResultTxt;
139:    delete ErrMsgTxt;
140:    delete Operand1Cmb;
141:    delete OperatorCmb;
142:    delete Operand2Cmb;
143:    delete ResultCmb;
144:    delete ErrMsgBox;
145:    delete CalcBtn;
146:    delete ExitBtn;
147: }
148:
149: int CAppWindow::OnCreate(LPCREATESTRUCT lpCS)
150: {
151:    int nX0= 20;
152:    int nY0= 30;
153:    int nX= nX0, nY= nY0;
```

```
154:    DWORD dwStaticStyle= WS - CHILD ©¦WS - VISIBLE ©¦SS - LEFT;
155:    DWORD dwEditStyle =  WS - CHILD ©¦WS - VISIBLE ©¦ WS - BORDER©¦
156:                    ES - LEFT ©¦ES - AUTOHSCROLL ©¦ES - UPPERCASE;
157:    DWORD dwComboStyle =  WS - CHILD ©¦WS - VISIBLE ©¦WS - VSCROLL;
158:    CRect r;
159:
160:    // create the first set of labels for the edit boxes
161:    makeRect(nX, nY, Wlbl, Hlbl, r);
162:    Operand1Txt= new CStatic();
163:    Operand1Txt- > Create("Operand1", dwStaticStyle, r, this);
164:    nX + = Wlbl+ LblHorzSpacing;
165:    makeRect(nX, nY, Wlbl, Hlbl, r);
166:    OperatorTxt= new CStatic();
167:    OperatorTxt- > Create("Operator", dwStaticStyle, r, this);
168:    nX + = Wlbl+ LblHorzSpacing;
169:    makeRect(nX, nY, Wlbl, Hlbl, r);
170:    Operand2Txt= new CStatic();
171:    Operand2Txt- > Create("Operand2", dwStaticStyle, r, this);
172:    nX + = Wlbl+ LblHorzSpacing;
173:    makeRect(nX, nY, Wlbl, Hlbl, r);
174:    ResultTxt= new CStatic();
175:    ResultTxt- > Create("Result", dwStaticStyle, r, this);
176:    // create the Operand1, Operator, Operand2, and Result
177:    // combo list boxes
178:    nX= nX0;
179:    nY + = Hlbl+ LblVertSpacing;
180:    makeRect(nX, nY, Wcmb, Hcmb, r);
181:    Operand1Cmb= new CComboBox();
182:    Operand1Cmb- > Create("dwComboStyle ©¦CBS - DROPDOWN, r,
183:                            this, ID - OPERAND1 - CMB);
184:    //create the Operator combo box
185:    nX + = Wcmb+ CmbHorzSpacing;
186:    makeRect(nX, nY, Wcmb, Hcmb, r);
187:    OperatorCmb= new CComboBox();
188:    OperatorCmb- > Create(dwComboStyle ©¦CBS - SIMPLE, r,
189:                            this, ID - OPERATOR - CMB);
190:    nX + = Wcmb+ CmbHorzSpacing;
191:    makeRect(nX, nY, Wcmb, Hcmb, r);
192:    operand2Cmb= new CComboBox();
193:    Operand2Cmb- > Create(dwComboStyle ©¦CBS - DROPDOWN, r,
194:                            this, ID - OPERAND2 - CMB);
195:    nX + = Wcmb+ CmbHorzSpacing;
196:    makeRect(nX, nY, Wcmb, Hcmb, r);
197:    ResultCmb= new CComboBox();
198:    ResultCmb- > Create(dwComboStyle ©¦CBS - DROPDOWN, r,
199:                            this, ID - RESULT - CMB);
200:    //create the static text and edit box for the error message
201:    nX= nX0;
202:    nY + = Hcmb+ CmbVertSpacing;
203:    makeRect(nX, nY, Wlbl, Hlbl, r);
204:    ErrMsgTxt= new CStatic();
205:    ErrMsgTxt- > Create(nX, nY, WLongbox, Hbox, r);
206:
207:
```

```
208:    ErrMsgBox= new CEdit();
209:    ErrMsgBox- > Create("Error Message", dwStaticStyle, r, this);
210:    //create the Calc push button
211:    nY + = Hbox+ BoxVertSpacing;
212:    makeRect(nX, nY, Wbtn, Hbtn, r);
213:    CalcBtn= new CxButon();
214:    CalcBtn- > Create("&Calc", r, this, ID - CALC - BTN);
215:    // Create the Exit Btn
216:    nX + =  Wbtn+ BtnHorzSpacing;
217:    makeRect(nX, nY, Wbtn, Hbtn, r);
218:    ExitBtn= new CxButton();
219:    ExitBtn- > Create ("&Exit", r, this, ID - EXIT - BTN);
220:    //clear the bInEror flag
221:    bInError= FALSE;
222:    return CFrameWnd::OnCreate(lpCS);
223: }
224:
225: void CAppWindow::InitAppWindow();
226: {
227:    char*  p[]= {"+ ","- ","* ","/","^ ",
228:                 "log","exp","sqrt","EOL"};
229:
230:    //add the operators in the Operator combo box
231:    for (int i= 0;p[i] ! = "EOL";i+ + )
232:    OperatorCmb- > AddString(p[i]);
233: }
234:
235: void   CAppWindow::HandleOperatorCmb()
236: {
237:    char s[MaxEditLen+ 1];
238:
239:    //get the text   in the Operator combo box edit area
240:    OperatorCmb- > GetWindowText(s, MaxEditLen);
241:    //use it to search for a matching list item
242:    OperatorCmb- > SelectString(- 1, s);
243: }
244:
245: void CAppWindow::HandleCalcBtn()
246: {
247:    int nOpIndex;
248:    double x, y, z;
249:    char opStr[MaxEidtLen+ 1];
250:    char s[MaxEditLen+ 1];
251:
252:    //obtain the string in the Operand1 combo box
253:    Operand1Cmb- > GetWindowText(s, MaxEditLen);
254:    //convert the string in the edit box area
255:    x =  atof(s);
256:    //obtain the string in the Operand2 combo box
257:    Operand2Cmb- > GetWindowText(s, MaxEditLen);
258:    //convert the string in the edit box area
259:    y =  atof(s);
260:    // obtain the string in the Operator combo box
261:    OperatorCmb- > GetWindwText (opStr, MaxEditLen);
```

```
262:    //clear the error message box
263:    ErrMsgBox- > SetWindowText("");
264:    bInError= FALSE;
265:    //determine the requested operation using the FindString
266:    //member function of CListBox
267:    nOpIndex= OperatorCmb- > FindString(- 1,opStr);
268:    switch(nOpIndex){
269:    case 0:// + operator
270:      z= x+ y;
271:      break;
272:    case 1:// - operator
273:      z= x- y;
274:      break;
275:    case 2:// * operator
276:      z= x* y;
277:      break;
278:    case 3:// /operator
279:      if(y ! = 0)
280:        z = x / y;
281:        else {
282:          z= 0;
283:          bInError= TRUE;
284:          ErrMsgBox- > SetWindowText("Division- by- zero error");
285:        }
286:        break;
287:    case 4:// ^ operator
288:        if (x> 0)
289:          z= exp(y* log(x));
290:        else {
291:          bInError= TRUE;
292:          ErrMsgBox- > SetWindowText(
293:            "cannot raise the power of a negative number");
294:        }
295:        break;
296:    case 5:// the natural logarithm function
297:        if (x> 0)
298:          z= log(x);
299:        else {
300:          bInError= TRUE;
301:            ErrMsgBox- > SetWindowText(
302:             "Invalid argument for the log(x) function");
303:        }
304:        break;
305:    case 6:// the exponential function
306:        if (x< 230)
307:          z= exp(x);
308:        else {
309:          bInError= TRUE;
310:          ErrMsgBox- > SetWindowText(
311:            "Invalid argument for the exp(x) function");
312:        }
313:        break;
314:    case 7:// the square root function
315:        if (x> = 0)
```

```
316:            z= sqrt(x);
317:          else {
318:            bInError= TRUE;
319:            ErrMsgBox- > SetWindowText(
320:              "Invalid argument for the sqrt(x) function");
321:          }
322:          break;
323:        default:
324:          bInError= TRUE;
325:          ErrMsgBox- > SetWindowText("Invalid operator");
326:          break;
327:     }
328:     //display the result if no error has occurred
329:     if (! bInError) {
330:        sprintf(s,"%g",z);
331:        ResultCmb- > SetWindowText(s);
332:        updateComboBox(ResultCmb);
333: }
334:    // update the operand combo boxes
335:    updateComboBox(Operand1Cmb);
336:    updateComboBox(Operand2Cmb);
337: }
338:
339:
340: void CAppWindow::makeRect(int nX,int nY,int nW,int nH,
341:                                    CRect &r)
342: {
343:    r.top= nY;
344:    r.left= nX;
345:    r.bottom= nY+ nH;
346:    r.right= nX+ nW;
347: }
348:
349: void CAppWindow::updateComboBox(CComboBox* pComboBox)
350: {
351:    char s[MaxEditLen+ 1]
352:    int nIdx;
353:
354:    pComboBox- > GetWinowText(s,MaxEditLen);
355:    //is string s in the combo list
356:    nIdx= pComboBox- > FindString(- 1,s);
357:    if (nIdx= = 0) return;
358:    else if (nIdx< 0) {
359:       pComboBox- > InsertString(0,s);
360:       // delete extra history list members
361:       while (pComboBox- > GetCount()> = MaxHistory)
362:          pComboBox- > DeleteString(pComboBox- > GetCount()- 1);
363:    }
364:    else {
365:      // delete the current selection
366:      pComboBox- > DeleteString(nIdx);
367:      // insert the string s at the first position
368:      pComboBox- > InsertString(0,s);
369:      // select the first combo box item
```

```
370:      pComboBox- > SetCurSel(0);
371:      }
372: }
373:
374: void CAppWindow::OnClose();
375: {
376:      if (MessageBox("Want to close this application",
377:                          "Query", MB - YESNO|MB - ICONQUESTION)= = IDYES)
378:        DestroyWindow();
379: }
380:
381: BEGIN - MESSAGE - MAP(CAppWindow, CFrameWnd)
382:       ON - BN - EDITUPDATE(ID - OPERATOR - CMB, HandleOperatorCmb)
383:       ON - BN - CLICKED(ID - CALC - BTN, HanleCalcBtn)
384:       ON - COMMAND(ID - CALC - BTN, CMCalcBtn)
385:      ON - COMMAND(ID - EXIT - BTN, HandleExitBtn)
386:       ON - BN - CLICKED(ID - EXIT - BTN, CMExitBtn)
387:       ON - COMMAND(CM - EXIT, OnExit)
388:       ON - WM - CREATE()
389:       ON - WM - CLOSE()
390: END - MESSAGE - MAP()
391:
392: // Construct the  CWindowApp's m - pMainWnd data member
393: BOOL CWindowApp::InitInstance()
394: {
395:    m - pMainWnd= new CAppWindw();
396:    m - pMainWnd- > ShowWindow(m - nCmdShow);
397:    m - pMainWnd- > UpdateWindow();
398:    return TRUE;
399: }
400:
401:  //application's constructor  initializes and runs the app
402: CWindowApp WindowApp;
```

分　析

清单 19.6 中的程序声明了三组常量。第一组常量用于指定各种应用程序控制的尺寸和间距,第二组常量声明了字符串的最大长度和选项的最大数目。第三组常量为各种控制建立 ID 号。

计算器应用程序声明了应用程序类 CWindowApp、窗口类 CAppWindow。以及按钮类 CxButton。CAppWindow 类声明了两组数据成员。第一个组为指向应用程序使用的各种控制的指针,第二组只含有 bInError 数据成员,用于标明一些计算错误。

CAppWindow 类声明了一个构造函数、一个析构函数和一组处理各种消息的成员函数,CAppWindow 构造函数执行如下任务:

- 加载加速键资源。
- 调用 Create 函数建立应用程序窗口(窗口是极大化的,并使用 EXITMENU 资源菜单)。
- 给数据成员 bInError 赋值为 FALSE。

· 调用 InitAppWindow 函数初始化应用程序。

第 149 到 223 行定义了成员函数 OnCreate, 该函数用于建立各种控制的实例。操作数和结果组合框建立时带有 CBS - DROPDOWN 风格, 这些组合框应避免含有 CBS - SORT 风格, 以便它们能作为历史列表框。函数用 CBS - SIMPLE 风格建立 Operator 组合框, 并保持相应列表框中的表项的有序性。

第 225 到 233 行定义了成员函数 InitAppWindow, 以初始化窗口实例, 将支持的运算符和数学函数加进 Operator 组合框的列表框组元中。这个过程使用了一个 for 循环, 在循环中调用了成员函数 AddString。

第 235 到 245 行定义了成员函数 HandleOperatorCmb, 以响应 Operator 组合框发出的 CBN - EDITUPDATE 通知消息。函数执行如下的任务:

· 使用 GetText 函数从编辑控制获得文本。
· 调用 SelectString 函数寻找与与所要检索的字符串匹配的表项(如果发现一个匹配 项, 匹配项作为选中文本出现在组合框的编辑控制区域中)。

第 245 到 337 行定义了成员函数 HandleCalcBtn, 以响应 Calc 按钮的通知消息并执行要求的计算。函数完成如下的系列操作:

· 检索 Operand1 组合框的编辑框中的字符串。
· 把检索的字符串转换成一个数值量, 并将其保存到局部变量 x 中。
· 检索 Operand2 组合框的编辑框中的字符串。
· 把检索的字符串转换成一个数值量, 并将其保存到局部变量 y 中。
· 获得 Operator 组合框的编辑控制中的字符串。
· 清除 Error Message 编辑控制, 将 FALSE 值赋给数据成员 bInError。
· 使用成员函数 CComboBox::FindString, 找到调用的运算符或数学函数在 Operator 组合框列表中的位置。
· 用一个 switch 语句确定要求的运算或函数计算, 随后执行要求的任务, 并把结果送给变量 z, 或标志一个错误(这是第一次在 COCA 程序中使用 switch 语句, 这种方式要比以往使用 strcmp 函数的嵌套 if 语句的版本, 运行要快一些)。
· 如没有错误发生, 在 Result 组合框的编辑控制中显示运算或函数计算结果, 并调用 updateComboBox 成员函数, 把新值插入 Result 组合框的列表框中。
· 把新操作数插入它们各自的组合框的列表框中, 两次调用 updateComboBox 成员函数, 更新两个操作域组合框(更新操作没有考虑出错条件, 而且, 如果要求计算一个数学函数, Operand2 组合框列表框并不改变, 除非在组合框中输入一个新的操作数。

函数 CMCalcBtn 响应 Alt-C 加速键消息, 此函数调用 HandleCalcBtn 成员函数来提供需要的响应;

第 349 到 372 行定义了成员函数 updatedComboxBox, 把编辑控制部分的字串插入列表框部分。此插入只在列表框部分没有这个字符串时发生。在本例中, 字符串插入到索引 0 的位置, 成为新的表头项。如果目标字符串已存在于列表框部分, 本函数将删除列表

框已存在的项,把它重插入索引 0 处。这样,目标字符串看上去是从原来位置移到列表框部分的顶部的。

HandleExitBtn 和 CMExitBtn 成员函数对 Exit 按钮和 Alt-E 加速键发出的消息作出同样响应。两个函数都发送 WM_CLOSE 消息给父窗口。

小　　结

今天这一课介绍了滚动条和组合框控制,这些控制都具有输入对象的共同特征。我们学习了如下的内容:

- 用滚动条控制可以从一个较大的整数范围中快速选择。
- 组合框控制有各种类型:简单的,下拉式和下拉列表式。可使用 CBN_EDITUPDATE 通知消息使组合框更加具有交互性能。
- 可将一个下拉组合框构成一个历史列表框。

问　与　答

问:滚动条只限于选择整数吗?

答:是。但是,这些整数可以是数组、列表框控制项的索引以及其他各种属性(例如颜色)的整数编码。因此,从某种意义上说,滚动条也可以用来选择非整数。

问:能建立一个在某一取值范围之外的滚动条控制吗?

答:不能。但可以使用一个列表框控制来代替,使控制列出每个不连续的成员的值。

作　　业

作业部分所提供的测验题有助于巩固理解所学内容,通过练习可以积累运用所学知识的经验。要尽量弄懂测验和练习的答案后再继续下一天的课程。附录 B 的"答案"中提供了解答。

测　验

1. 对还是错?

 如果在建立一个组合列表框时没有包括 CBS_AUTOHSCROLL 风格,则文本字符被限制在编辑控制的矩形边界里。

2. 对还是错?

 可以处理 CBN_SELCHANGE 通知消息,以便监视组合框的编辑控制的每次按键。

3. 对还是错?

 带有 CBS_SORT 风格建立的组合框,它的表项是唯一的和排好序的。

4. 对还是错?

用于仿真一个历史列表框的组合框，建立时不能带有 CBS_SORT 风格。

5. 对还是错?

 历史列表可以有重复项。

6. 对还是错?

 CTLSCRL1.CPP 示意义了怎样将一个滚动条控制的当前值与一个文本框的数值联系起来。

7. 对还是错?

 建立控制时，滚动条控制的范围是固定的。

练　习

1. 修改 CTLCOMB1.CPP 程序，加入一个 Variables 多行框和一个 Store 下压按钮。
2. 修改 CTLCOMB1.CPP 程序，增加一个 Variables 列表框和一个 Store 下压按钮。

第20天

对 话 框

对话框是一种特殊的子窗口,它含有用于显示信息或接收数据的控制。Windows 应用程序使用对话框和用户交换信息。今天这一课,我们将学习 Windows 所支持的模式和无模式对话框。

新 词

模式对话框(modal dialog box) **无模式对话框**(modeless dialog box) 对于模式对话框,只有在关掉它以后,才能让应用程序执行下一步的任务,因为他们往往意味着执行的是至关重要的数据交换。事实上,模式对话框获得输入焦点后会禁止父窗口。无模式对话框则无需关闭便可继续使用应用程序。

今天,我们将学习下列内容:

- 构造类 CDialog 的实例
- 执行模式对话框
- 传送控制数据
- 传送模式对话框数据
- 传送无模式对话框数据

20.1 构造对话框

MFC 库中声明了支持模式和无模对话框的 CDialog 类。CDialog 是 CWnd 的子孙类,它有一个构造函数和若干成员函数,成员函数中包括 Create 函数。CDialog 构造函数是作为保护成员声明的,因此使用者必须建立自己的无模式对话框类。这些派生类可以声明以下两组成员中的任一组以补充自定义的对话框类:

- 只声明一个引用 CDialog::Create 成员函数的公有的构造函数。用这种方法建立对话框实例只需一步。
- 声明一个公用的构造函数和一个 Create 成员函数,这种方法用两个步骤建立对话框实例:首先调用类构造函数,随后调用 Create 成员函数。

重载的成员函数 CDialog::Create 声明如下:

```
BOOL Create(LPCSTR lpTemplateName, CWnd * pParentWnd = NULL);
BOOL Create(UINT nIDTemplate,CWnd * pParentWnd = NULL);
```

参数 lpTemplateName 和 nIDTemplate 分别为对话框的资源名和 ID。参数

pParentWnd 为指向父窗口的指针。

当对话框实例也是应用程序主窗口时,可以使用缺省值。

> **注　意**　使用资源定义对话框和其中的控制,就可以在 Windows 应用程序源代码外,定义一个控制的位置、大小、风格和标题等。这样,就可以改变一个资源文件,重编译,然后再加入到.EXE 应用程序文件中,不必重新编译源代码。这种方法,只需应用程序代码的一个拷贝,就可以开发出具有不同的颜色、风格、以及语言的不同资源版本。

> **注　意**　Visual C+ + 软件包中的 App Studio,允许以 Visual Basic 风格通过在对话框中绘制控制建立对话框。App Studio 建立的.RC 资源文件,可以并入 Windows 应用程序中。如果你是 Windows 编程的新手,首先应学习.RC文件及其描述。App Studio 的使用是非常容易和直观的,若了解有关.RC 资源描述方面的知识,使用 App Studio 生成资源就更加容易。

20.2　执行模式对话框

一般来说,建立和撤消最频繁的是模式对话框,其次是无模式对话框,再次才是窗口。执行模式对话框的步骤如下:

1. 用对话框类的构造函数建立一个对话框实例。
2. 调用在类 CDialog 中声明的 DoModal()成员函数,生成对话框。通常,对话框中包含 OK 和 Cancel 下压按钮,其中 OK 按钮为缺省的。OK 和 Cancel 按钮所对应的预定义 ID 值分别为 IDOK 和 IDCANCEL。下压按钮可以用不同于 OK 和 Cancel 的标题,但是仍应使用 IDOK 和 IDCANCEL 处理更名后的按钮。使用这些 ID,就可利用对话框中定义的成员函数 OnOk 和 OnCancel,以自动响应 IDOK 和 IDCANCEL。单击 OK 或按下 Enter 键通常表明承认了对话框中当前的(缺省的或编辑过的)数据。与此相反,单击 Cancel 表明不满意当前数据。函数 DoModal()声明如下:

   ```
   int DoModal();
   ```

 函数返回一个代表结果的整数。通常的结果值为 IDOK 或 IDCANCEL。
3. 把 DoModal 函数的结果与 IDOK(很少使用 IDCANCEL)比较。根据比较的结果决定采取的步骤,通常要涉及到存取键入对话框控制的数。

OnInitDialog(从类 CDialog 继承而来)、OnOK 和 OnCancel 成员函数支持模式对话框的执行。因为这些无参数函数声明为虚函数,因此这些重载函数应避免使用参数。成员函数 OnInitDialog 用于初始化对话框及其控制。OnInitDialog 声明如下:

```
virtual BOOL OnInitDialog();
```

函数返回一个布尔型结果,表明对话框的初始化是否成功。通常,OnInitDialog 函数初始化对话框的控制,初始化动作通常涉及到从缓冲区或数据成员中拷贝数据。

OnOK 成员函数处理按下 OK 按钮事件。OnOK 函数的声明如下:

```
virtual void OnOK();
```

OnOK 函数负责把对话框控制的数据拷到数据成员或缓冲区中。通过使用 EndDialog 成员函数,OnOK 函数可以给 DoModal 函数提供它的返回值。通常,函数 OnOK 定义中,最后一条语句是 EndDialog(IDOK),后一个函数调用使 DoModal 函数返回 IDOK。

OnCancel 成员函数处理按下 Cancel 按钮事件。函数 OnCancel 声明如下:

```
virtual void OnCancel();
```

OnCancel 函数负责在对话框关闭之前的清扫工作,如关闭所用到的数据文件。通常,OnCancel 成员函数定义中的最后一条语句是 EndDialog(IDCANCEL)。后一个函数调用使 DoModal 函数返回 IDCANCEL。

以下是一个简单的 MFC 程序的例子。其中使用了一个在资源文件中定义的对话框。该程序还使用资源文件建立了同一对话框的两种可选形式——一种使用现代英语,另一种使用古英语。应用程序比较简单,仅由一个带单个菜单项 Exit 的空窗口组成。当单击 Exit 菜单项(或按 Alt-X 键)时,会出现一个对话框,询问是否想退出应用程序。这个对话框有一个标题、一条消息和两个按钮(实际上,CWnd::MessageBox 函数会产生类似的对话框)。程序可以从两个对话框版本中选择,当第一次选中 Exit 菜单时,显示的是现代英语版本(带 OK 和 Cancel 按钮),如图 20.1 所示。如果你单击 Cancel 按钮,再选中 Exit 菜单,则出现的是对话框的古英语版本(带 Yea 和 Nay 按钮),如图 20.2 所示。每次选中 Cancel 或 Nay 按钮,并选中 Exit 菜单,便可在对话框的两个版本中切换。退出应用程序,根据当前对话框版本,单击 OK 或 Yea 按钮。

清单 20.1 列出了头文件 DIALOG1.H 的源代码;清单 20.2 为资源描述文件 DIALOG1.RC 的代码;清单 20.3 列出 DIALOG1.CPP 程序的源代码。

键　入　　清单 20.1　头文件 DIALOG1.H 的源代码

```
1:# define CM_EXIT (WM_USER + 100)
```

键　入　　清单 20.2　资源文件 DIALOG1.RC 的源代码

```
1: # icnlude < windows.h>
2: # include < afxres.h>
3: # include "dialog1.h"
```

图 20.1　用现代英语单词显示对话框的应用程序 DIALOG1.EXE 的运行范例

图 20.2　用古英语单词显示对话框的应用程序 DIALOG1.EXE 的运行范例

```
4:
5: EXITMENU MENU LOADONCALL MOVEABLE PURE DISCARDABLE
6: BEGIN
7: MENUITEM "E&xit", CM_EXIT
8: END
9:
```

```
10: NEW DIALOG DISCARDABLE LOADONCALL PURE MOVEABLE 30, 50, 200, 100
11: STYLE WS- POPUP©¦DS- MODALFRAME
12: CAPTION "Message"
13: BEGIN
14: CTEXT"Exit the application?", 1, 10, 10, 170, 15
15: CONTROL "OK", IDOK, "BUTTON", WS- CHILD ©¦WS- VISIBLE©¦
16: WS- TABSTOP ©¦BS- DEFPUSHBUTTON, 20, 50, 70, 15
17: CONTROL "Cancel", IDCANCEL, "BUTTON", WS- CHILD ©¦WS- VISIBLE©¦
18: WS- TABSTOP ©¦BS- PUSHBUTTON, 110, 50, 70, 15
19: END
20:
21: OLD DIALOG DISCARDABLE LOADONCALL PURE MOVEABLE 30, 50, 200, 100
22: STYLE WS- POPUP ©¦DS- MODALFRAME
23: CAPTION " Word With Thou"
24: BEGIN
25: CTEXT "Thou leavest now?" 1, 10, 10, 170, 15
26: CONTROL "Yea", IDOK, "BUTTON", WS©¦CHILD ©¦WS- VISIBLE ©¦
27: WS- TABSTOP ©¦BS- DEFPUSHBUTTON, 20, 50, 70, 15
28: CONTROL "Nay", IDCANCEL, "BUTTON", WS- CHILD ©¦WS- VISIBLE ©¦
29: WS- TABSTOP ©¦BS- PUSHBUTTON, 110, 50, 70, 15
30: END
```

键　入　　**清单 20.3　程序文件 DIALOG1.CPP 的源代码**

```
1: # include < afxwin.h>
2: # include "dialog1.h"
3:
4: class CWindowApp：  public CWinApp
5: {
6: public:
7: virtual BOOL InitInstance();
8: };
9: // expand the functionality of CFrameWnd by deriving
10: // class CAppWindow
11: class CAppWindow：  public CFrameWnd
12: {
13: public:
14:
15: CAppWindow();
16:
17: protected:
18:   // handle exiting the application
19:   afx- msg void OnExit()
20:         { SendMessage(WM- CLOSE); }
21:   // Handle closing the window
22:   afx- msg void OnClose();
23:
24:   DECLARE- MESSAGE- MAP();
25:
```

```
26: };
27:
28: CAppWindow::CAppWindow()
29: {
30:   // create the window
31:   Create(NULL, "A Bare- Bones Dialog Box Tester",
32:             WS- OVERLAPPEDWINDOW, rectDefault, NULL,"EXITMENU");
33:
34: }
35:
36: void CAppWindow::OnClose()
37: {
38:   static BOOL bFlag= FALSE;
39:   CDialog*  pDialogBox;
40:   BOOL bCloseIt;
41:
42:   // toggle flag that selects alternate dialog box resources
43:   bFlag= (bFlag= = TRUE) ? FALSE: TRUE;
44:   if (bFlag) {
45:        pDialogBox= new CDialog("NEW",this);
46:        // use Modern English dialog box
47:        bCloseIt= (pdialogBox- > DoModal()= = IDOK) ?TRUE:  FALSE;
48:    }
49:    else {
50:        pdialogBox= new CDialog("OLD",this);
51:        // use Old English dialog box
52:        bCloseIt= (pDialogBox- > DoModal()= = IDOK) ?TRUE:  FALSE;
53:    }
54:   if (bCloseIt)
55:        DestroyWindow();
56: }
57:
58: BEGIN- MESSAGE- MAP(CAppWindow,CFrameWnd)
59: ON- COMMAND(CM- EXIT,OnExit)
60: ON- WM- CLOSE()
61: END- MESSAGE- MAP()
62:
63: // Construct the CWindowApp's m- pMainWnd data member
64: BOOL CWindowApp::InitInstance()
65: {
66:    m- pMainWnd= new CAppWindow();
67:    m- pMainWnd- > ShowWindow(m- nCmdShow);
68:   m- pMainWnd- > UpdateWindow();
69:   return TRUE;
70: }
71:
72: // application's constructor initializes and runs the app
73: CWindowApp WindowApp;
```

分　析

清单 20.2 中列出了资源描述文件 DIALOG1.RC,其中定义了如下的资源:

- 菜单资源 EXITMENU,显示一个菜单,其中只有一个选项 Exit。
- 对话框资源 NEW,定义了风格、标题和一组子控制。指定的风格表明,这个对话框是模式弹出式子窗口。指定的标题为字符串"Message"。对话框包含三种控制:居中的静态文本(用于对话框消息)、缺省的 OK 按钮以及一个普通的 Cancel 按钮。OK 按钮的资源 ID 为预定义常量 IDOK,Cancel 按钮的资源 ID 为预定义的常量 IDCANCEL。
- 对话框资源 OLD,它除了使用古英语词汇这一点外,其余与对话框资源 NEW 很相似。Yea 按钮的资源 ID 为预定义常量 IDOK,Nay 按钮的资源 ID 为预定义的常量 IDCANCEL。这些按钮是带一个典型标题的退出按钮的范例。

对话框和控制的资源定义对你来说可能比较陌生。附录 A 更加详细地讨论了用于对话框及其控制的资源文件。关键字 CTEXT 指定文本居中,关键字 CONTROL 允许定义任意控制,并要求具有控制的标题,ID,控制类,控制风格,位置和大小。

清单 20.3 中列出了 DIALOG1.CPP 程序文件的源代码。源代码声明了两个类:一个应用程序类和一个窗口类,应用程序使用 CDialog 类,是因为没要求附加对话框功能。

我们最关心的成员函数为 CAppWindow::OnClose,它负责处理 Exit 菜单项发出的 WM_CLOSE 命令消息。本函数使用布尔型静态局部变量 flag 在新、旧两个对话框中轮流切换。新英语对话框由如下的语句产生:

```
pDialogBox= new CDialog("NEW",this);
```

对话框使用 DoModal 函数来执行,且在单击某个下压按钮控制之前,父窗口被禁止。成员函数 DoModal 返回的值将与常量 IDOK 比较,根据结果给结局域布尔型变量 bCloseIt 赋值 TRUE 或 FALSE。函数 OnClose 检查 bCloseIt 的值,断定是否关闭应用程序窗口。

对话框的古英语版本实例的建立与此相类似,其中使用了如下的语句:

```
pDialogBox= new CDialog("OLD",this);
```

CDialog::OnOK 成员函数在响应 IDOK 命令时,关闭对话框并返回 IDOK 值。

20.3　传送控制数据

对话框主要是以弹出式窗口的形式要求应用程序用户输入数据,这种输入经常包括使用圆按钮、选中框和编辑框进行各种设置。因为对话框的建立频繁,因而很有必要为其下一次出现保持对话框控制中的数据。在许多 Windows 编辑器中出现的 Search 和 Replace 对话框就是典型的例子。这些对话框记住上次执行它们时其中的所有或部分控制的设置。

使用对话框的这种特性需要有一种在对话框和缓冲区之间进行数据传输的机制。这类缓冲区通常是父窗口的一个数据成员。支持数据传送的第一步是定义一个传送缓冲区类型。缓冲区声明的数据域是用来对传输数据的控制进行缓冲，这些控制通常包括静态文本、编辑框、列表框、组合框、滚动条、选中框和圆按钮。分组框和下压按钮控制没有数据传送，因而数据传送缓冲区类型的声明中没有它们。以下是一个数据缓冲区类型的例子，其中包含所有允许的控制的实例：

```
struct TAppTransferBuffer {
char StaticText[MaxStaticLen];
char EditBox[MaxEditLen];
CListBoxData ListBoxData; // user- defined class
CComboBoxData ComboBoxData; // user- defined class
CScrollBarData ScrollBarData; // user- defined class
int CheckBox;
int RadioButton;
};
```

缓冲区结构中只需包括参与数据传送的控制，而不必以任何特定的顺序声明缓冲区结构中的域。以上的缓冲区类型举例中有三种特殊的用户定义的类，分别用于在对话框与列表框、组合框和滚动条之间进行数据传送。本节随后部分有关于这些类的详述。

> **注　意**　MFC 库中声明了类 CDataExchange，用于支持 MFC 库所使用的对话数据交换和对话数据验证例程。可以使用 CDataExchange 类来代替本书介绍的方法。有关 CDataExchange 的更详细的内容，请查阅\MSVC\HELP\MFCNOTES.HLP 的 Technical Note 26。

我们来看看数据传送缓冲区类型的各个成员：

- Static Text 成员协助在静态文本控制和数据缓冲区之间传送数据。数据成员定义了一个比静态文本控制字符数多或相等的字符数组。
- EditBox 成员辅助在编辑框控制和数据缓冲区之间的数据移动、数据成员定义了一个其长度大于或等于编辑框控制中的字符数的字符数组。
- ListBoxData 成员协助在列表框控制和数据缓冲区之间进行数据传送。ListBoxData 是声明用来管理对话框和列表框之间数据传送的类的一个实例。为什么使用专门的数据传送类？这样的类可以成功地封装数据和执行一些有关传送数据的操作。CListBoxData 类通常声明为：

```
class CListBox
{
public:
  // array of CString to store the list box items
  CStringArray DataList;
  CListBoxData();
  // transfer data to the buffer
  void GetListBoxData(CListBox* pListBox);
  // transfer data to the list box
```

```
    void SetListBoxData(CListBox* pListBox);
};
```

成员 DataList 是 CString 对象的一个动态数组，用于存储列表项。MFC 库声明了 CStringArray 类并提供充足的对字符串数组的操作。因为类 CStringArray 提供了 GetSize 成员函数以获得 CStringArray 实例中字符串对象的数目，所以缺少一个用于跟踪列表框项数的数据成员函数亦无妨。

成员函数 GetListBoxData 和 SetListBoxData 提供了在列表框控制和数据缓存区之间的双向数据传送。

- 成员 ComboBoxData 协助在组合框和数据缓冲区之间进行数据移动。ComboBoxData 是 CComboBoxData 类的一个实例，以下是类 CComboBoxData 的一个典型声明：

```
class CComboBox
{
public:
    // the combo box selection
    CString Selection;
    // array of CString to store the list box items
    CStringArray DataList;
    CComboBoxData();
    // transfer data to the buffer
    void GetComboBoxData(CListBox* pComboBox);
    // transfer data to the combo box
    void SetComboBoxData(CListBox* pComboBox);
};
```

类 CComboBoxData 的声明与类 CListBoxData 相类似。类 CComboBoxData 有一个附加数据成员 Selection，用于存储组合框选项在编辑控制中的位置。

- 成员 ScrollBarData 协助在滚动条和数据缓冲区之间进行数据传送。该成员中包含一个 CScrollBarData 类，定义如下：

```
class CScrollBarData
{
public:
    int LowValue;
    int HighValue;
    int Position;
    CScrollBarData();
    // move data to the buffer
    void GetScrollBarData(CScrollBar* pScrollBar);
    // move data to the scroll bar
    void SetScrollBarData(CScrollBar* pScrollBar);
};
```

成员 LowValue, HighValue 和 Position 保存滚动条范围和当前游标位置。

- CheckBox 成员保存表示选中框的当前选中状态的一个整数值。
- RadioButton 成员保存表示圆按钮当前选中状态的一个整数值。

20.4　模式对话框的数据传送

下面的应用程序是模式对话框控制与缓冲区之间进行数据传送的简单示范。它建立一个用于替换文本编辑器中的字符的典型的对话框。对话框中包含如下的控制:

- Find 编辑框
- Replace 编辑框
- Scope 分组框, 其中包括 Global 和 Selected Text 圆按钮控制
- Case- Sensitive 选中框
- Whole Word 选中框
- OK 下压按钮控制
- Cancel 下压按钮控制

应用程序的主菜单有 Exit 和 Dialog 两个菜单项, 单击 Dialog 菜单项或按 Alt-D 键就可调用对话框, 当第一次调用对话框, 其中的控制有如下的初始值及状态:

- Find 编辑框包含字符串 DOS
- Replace 编辑框包含字符串 Windows
- Global 圆按钮被选中
- Case-Sensitive 选中框被选中
- Whole Word 选中框被选中

在编辑框中键入新字符串, 改变圆按钮和选中框的选中状态。现在, 单击 OK 按钮(或按 Alt-O 键), 关闭对话框, 再次选 Dialog 菜单项以弹出对话框。应注意到, 对话框控制的值和状态与上次关闭时一样。

清单 20.4 列出了头文件 DIALOG2.H 源代码; 清单 20.5 为资源描述文件 DIALOG2.RC 的代码; 清单 20.6 为程序文件 DIALOG2.CPP 的源代码。

键　入　　清单 20.4　头文件 DIALOG2.H 的源代码

```
1: # define CM_EXIT (WM_USER+ 100)
2: # define CM_DIALOG (WM_USER+ 101)
3:
4: # define ID_FIND_TXT 201
5: # define ID_FIND_EDIT 202
6: # define ID_REPLACE_TXT 203
7: # define ID_REPLACE_EDIT 204
8: # define ID_SCOPE_GRP 205
9: # define ID_GLOBAL_RBT 206
10: # define ID_SELTEXT_RBT 207
11: # define ID_CASE_CHK 208
12: # define ID_WHOLEWORD_CHK 209
```

键　入　　**清单 20.5　资源描述文件 DIALOG2.RC 的代码**

```
1: # include < windows.h>
2: # include < afxres.h>
3: # include "dialog2.h"
4:
5: BUTTONS ACCELERATORS
6: BEGEIN
7: "o" IDOK, ALT
8: "C", IDCANCEL, ALT
9: END
10:
11: SEARCH DIALOG DISCARDABLE LOADONCALL PURE MOVEABLE 10, 10, 200, 150
12:
13: STYLE WS- POPUP ©¦WS- CLIPSIBLINGS ©¦WS- CAPTION ©¦WS- SYSMENU ©¦
14:         DS- MODALFRAME
15: CAPTION "Controls Demo"
16: BEGIN
17:   CONTROL "Find", ID- FIND- TXT, "STATIC", WS- CHILD ©¦WS- VISIBLE ©¦
18:   SS- LEFT, 20, 10, 100, 15
19:   CONTROL " ", ID- FIND- EDIT, "EDIT", WS- CHILD ©¦WS- VISIBLE ©¦
20:   WS- BORDER ©¦WS- TABSTOP, 20, 25, 100, 15
21:   CONTROL "Replace", ID- REPLACE- TXT, "STATIC", WS- CHILD ©¦
22:   WS- VISIBLE ©¦SS- LEFT, 20, 45, 100, 15
23:   CONTROL " ", ID- REPLACE- EDIT, "EDIT", WS- CHILD ©¦WS- VISBILE ©¦
24:   WS- BORDER ©¦WS- TABSTOP, 20, 60, 100, 15
25:   CONTROL "Scope", ID- SCOPE- GRP, "BUTTON", WS- CHILD ©¦
26:   WS- VISIBLE ©¦WS- TABSTOP ©¦BS- GROUPBOX, 20, 80, 90, 50
27:   CONTROL "Global", ID- GLOBAL- RBT, "BUTTON", WS- CHILD ©¦
28:   WS- VISIBLE ©¦WS- TABSTOP ©¦BS- AUTORADIOBUTTON, 30, 90, 50, 15
29:   CONTROL "Selected Text", ID- SELTEXT- RBT, "BUTTON", WS- CHILD ©¦
30:   WS- VISIBLE ©¦WS- TABSTOP ©¦BS- AUTORADIOBUTTON, 30, 105, 60, 15
31:   CONTROL "Case Sensitive", ID- WHOLEWORD- CHK, "BUTTON", WS- CHID ©¦
32:   WS- VISIBLE ©¦WS- TABSTOP ©¦BS- AUTOCHECKBOX, 20, 130, 80, 15
33:   CONTROL "Whole Word", ID- WHOLEWORD- CHK, "BUTTON"", WS- CHILD ©¦
34:   WS- VISIBLE ©¦WS- TABSTOP ©¦BS- AUTOCHECKBOX, 20, 130, 80, 15
35:   CONTROL, "&OK", IDOK, "BUTTON", WS- CHILD ©¦WS- VISIBLE ©¦
36:   WS- TABSTOP ©¦WS- GROUP ©¦BS- DEFPUSHBUTTON, 120,  90, 30, 20
37:   CONTROL"&Cancel", IDCANCEL, "BUTTON", WS- CHILD ©¦WS- VISIBLE ©¦
38:   WS- TABSTOP ©¦WS- GROUP ©¦BS- PUSHBUTTON, 160, 90, 30, 20
39: END
40:
41: MAINMENU MENU LOADONCALL MOVEABLE PURE DISCARDABLE
42: BEGIN
43:   MENUITEM "E&xit", CM- EXIT
44:   MENUITEM "&Dialog", CM- DIALOG
45: END
```

键　入　　清单 20.6　程序文件 DIALOG2.CPP 的源代码

```
1: # include < string.h>
2: # include < afxwin.h>
3: # include "dialog2.h"
4:
5: // declare constants
6: const MaxEditLen= 30;
7: const BF_CHECKED= 1;
8: const BF_UNCHECKED= 0;
9: // declare the application buffer
10: struct TAppTransferBuf {
11: char FindBoxBuff[MaxEditLen+ 1];
12: char ReplaceBoxBuff[MaxEditLen+ 1];
13: int GlobalRbtBuff,
14: int SelTextRbtBuff,
15: int CaseChkBuff;
16: int WholeWordChkBuff,
17: };
18:
19: class CWindowApp：  public CWinApp
20: {
21: public:
22:   virtual BOOL InitInstance();
23: };
24:
25: // expand the functionality of CDialog by
26: // deriving class CAppDialog
27: class CAppDialog：  public CDialog
28: {
29: public:
30:
31:   CAppDialog(const char FAR*  lpTemplateName,
32:                    CWnd*  pParentWnd= NULL):
33:   CDialog(lpTemplateName, pParentWnd) {}
34:
35: protected:
36:
37:     // handle initializing the dialog box
38:    virtual BOOL OnInitDialog();
39:    // handle pressing the OK button
40:    virtual void OnOK();
41:    // pointer to data buffer in parent window
42:    TAppTransferBuf*  pAppBuffer;
43: };
44:
45:     // expand the functionality of CFrameWnd by deriving class CAppWindow
46: class CAppWindow：  public CFrameWnd
47: {
48:
```

```
49: friend CAppDialog;
50:
51: public:
52:
53:    // declare data transfer buffer
54:     TAppTransferBuf AppBuffer;
55:
56:    CAppWindow();
57:    // handle the dialog command
58:  afx_msg void CMDialog();
59:  // handle exiting the application
60:  afx_msg void OnExit()
61:  { SendMessage(WM_CLOSE); }
62:
63: protected:
64:
65:  // handle closing the window
66:  afx_msg void OnClose();
67:
68:  // declare message map macro
69:  DECLARE_MESSAGE_MAP();
70:
71: };
72:
73: BOOL CAppDialog::OnInitDialog()
74: {
75:  CAppWindow* pW= (CAppWindow* ) (GetParent());
76:  pAppBuffer= &pW-> AppBuffer;
77:  // copy data from the buffer to the dialog box controls
78:  SetDlgItemText(ID_FIND_EDIT,pAppBuffer-> FindBoxBuff);
79:  SetDlgItemText(ID_REPLACE_EDIT,pAppbuffer-> ReplaceBoxBuff);
80:  CButton* pBtn= (CButton* )(GetDlgItem(ID_GLOBAL_RBT));
81:  pBtn-> SetCheck(pAppBuffer-> GlobalRbtBuff);
82:  pBtn= (CButton* ) (GetDlgItem(ID_SELTEXT_RBT));
83:  pBtn-> SetCheck(pAppBuffer-> SelTextRbtBuff);
84:  pBtn= (CButton* ) (GetDlgItem(ID_CASE_CHK));
85:  pBtn-> SetCheck(pAppBuffer-> CaseChkBuff);
86:  pBtn= (CButton* ) (GetDlgItem(ID_WHOLEWORD_CHK));
87:  pBtn-> SetCheck(pAppBuffer-> WholeWordChkBuff);
88:  return TRUE;
89: }
90:
91: void CAppDialog::OnOK()
92: {
93:  // copy data from the dialog box controls to the buffer
94:  GetDlgItemText(ID_FIND_EDIT,pAppBuffer-> FindBoxBuff,
95:  MaxEditLen);
96:  GetDlgItemText(ID_REPLACE_EDIT,pAppBuffer-> ReplaceBoxBuff,
97:  MaxEditLen);
98:  CButton* pBtn= (CButton* ) (GetDlgItem(ID_GLOBAL_RBT));
99:  pAppBuffer-> GlobalRbtBuff= pBtn-> GetCheck();
100:  pBtn= (CButton* )(GetDlgItem(ID_SELTEXT_RBT));
```

```
101:    pAppBuffer->SelTextRbtBuff= pBtn->GetCheck();
102:    pBtn= (CButton* ) (GetDlgItem(ID_CASE_CHK));
103:    pAppBuffer->CaseChkBuff= pBtn->GetCheck();
104:    pBtn= (CButton* ) (GetDlgItem(ID_WHOLEWORD_CHK));
105:    pAppBuffer->WholeWordChkBuff= pBtn->GetCheck();
106:     // return dialog box value
107:    EndDialog(IDOK);
108: }
109:
110: CAppWindow::CAppWindow()
111: {
112:    // load the accelerator keys resource
113:    LoadAccelTable("BUTTONS");
114:
115:    // create the window
116:    Create(NULL, "Modal Dialog Box Data Transfer Tester",
117:          WS_OVERLAPPEDWINDOW, rectDefault, NULL, "MAINMENU");
118:
119:    // fill buffer with 0's
120:    memset(&AppBuffer, 0x0, sizeof(AppBuffer));
121:    strcpy(AppBuffer.FindBoxBuff, "Windows");
122:    strcpy(AppBuffer.ReplaceBoxBuff, "Windows");
123:    AppBuffer.GlobalRbtBuff= BF_CHECKED;
124:    AppBuffer.CaseChkBuff= BF_CHECKED;
125:    AppBuffer.WholeWordChkBuff= BF_CHECKED;
126: };
127:
128: void CAppWindow::CMDialog()
129: {
130:    CString msgStr;
131:    CAppDialog*  pDlg= new CAppDialog("SEARCH", this);
132:
133:    if (pDlg->DoModal()= = IDOK) {
134:          msgStr= "Find String: ";
135:          msgStr + = AppBuffer.FindBoxBuff;
136:          msgStr + = "\n\nReplace String: ";
137:          msgStr + = AppBuffer.ReplaceBoxBuff;
138:          MessageBox((const char* )msgStr, "Dialog Box Data", MB_OK);
139:    }
140: }
141:
142: void CAppWindow::OnClose()
143: {
144:     if (MessageBox("Want to close this application",
145:                         "Query", MB_YESNO ©¦MB_ICONQUESTION)= = IDYES)
146:      DestroyWindow();
147: }
148:
149: BEGIN_MESSAGE_MAP(CAppWindow, CFrameWnd)
150:      ON_COMMAND(CM_DIALOG, CMDialog)
151:     ON_COMMAND(CM_EXIT, OnExit)
152:      ON_WM_CLOSE()
```

```
153: END_MESSAGE_MAP()
154:
155: // Construct the CWindowApp's m_pMainWnd data member
156: BOOL CWindowApp::InitInstance()
157: {
158:    m_pMainWnd= new CAppWindow();
159:    m_pMainWnd->ShowWindow(m_nCmdShow);
160:    m_pMainWnd->UpdateWindow();
161:    return TRUE;
162: }
163:
164: // application's constructor initializes and runs the app
165: CWindowApp WindowApp;
```

图 20.3 为程序 DIALOG2.EXE 的一个运行范例。

图 20.3　程序 DIALOG2.EXE 的一个运行范例

分　析

清单 20.5 中列出了资源描述文件 DIALOG2.RC,该文件定义了加速键、菜单和对话框及其控制的资源。在对话框资源定义中,OK 和 Cancel 下压按钮的资源 ID 值分别为预定义的 IDOK 和 IDCANCEL。OK 按钮具有缺省下压按钮风格,Cancel 按钮具有正常下压按钮风格。分组框、圆按钮和选中框具有 BUTTON 类名。与这些控制联系的风格决定它们的最终形态。附录 A 给出了有关声明对话框所属控制的资源的更详细说明。

清单 20.6 为程序文件 DIALOG2.CPP 的源代码。程序声明了数据传送类型 TAppTransferBuf,其中包含了用于编辑框、圆按钮和选中框的成员。

程序清单声明三个类:一个应用程序类、一个窗口类和一个模式对话框类。应用程序的对话框类 CAppDialog 声明了一个类构造函数、一些成员函数和数据成员 pAppBuffer。数据成员 pAppBuffer 为指向数据缓冲区 AppBuffer 的指针。这是一个用于管理数据缓冲区的数据隐藏方案。若把传送缓冲区当一个全局变量声明,则可简化代码。记住,这种方法使缓冲区可被一个庞大的 Windows 应用程序的所有种类的对象存取,因而极易受破坏。类的构造函数调用其父类构造函数建立实例。

第 73 到 89 行中定义了成员函数 OnInitDialog 成员函数,该函数执行以下的任务来初始化对话框:

- 把父窗口地址赋给局部指针 pW。
- 使用 pW 指针,使数据成员 pAppBuffer 指向父窗口的成员 AppBuffer。
- 从缓冲区获得数据,送到模式对话框的控制中。其中使用了函数 CWnd::SetDlgItemText 和用于存取不同下压按钮控制的局部指针 pBtn。(pBtn 和函数 SetCheck()设置各种按钮和选中框控制的选中状态)。

第 91 到 108 行定义了 OnOK 成员函数,把数据从对话框移到数据缓冲区中。其中使用了函数 CWnd::GetDlgItemText 和一指向下压按钮控制的局部指针 pBtn(指针 pBtn 和函数 GetCheck()获得各种圆按钮和选中框控制的选中状态)。

应用程序窗口类 CAppWindow 声明了一个公用数据传送缓冲区 AppBuffer、一个类构造函数和三个成员函数,同时声明类 CAppDialog 为友元。若要让对话框成员函数存取 AppBuffer 成员,可以使 AppBuffer 成为公有成员并且把 CAppDialog 声明为友元。

CAppWindow 构造函数建立窗口(也装入菜单资源),然后初始化数据缓冲区。调用 memset 函数,以零填充成员 AppBuffer,在赋一个特定值给控制之前,应先调用 memset 函数执行缓冲区的基本初始化。构造函数随后分别把"DOS"和"Windows"字符串赋给 Find 和 Replace 编辑框的缓冲区。构造函数也把 BF_CHECKED 赋给 GlobalRbtBuff,CaseChkBuff 和 WholeWordChkBuf 成员。

CAppWindows 类在第 128 到 140 行定义了成员函数 CMDialog,用于处理 Dialog 菜单发出的命令消息。本函数使用 DoModal 成员函数引用一个对话框实例,并把函数的返回结果与 IDOK 比较。如果两项一致,CMDialog 函数建立和显示一条反映当前 Find 和 Replace 文本的消息字符串。

20.5　传送无模式对话框的数据

本节提供了一个传送无模式对话框数据的 MFC 应用程序。程序演示了怎样建立一个无模式对话框和怎样传送特定下压按钮控制使用的数据。接下来的应用程序使用了 DIALOG2.CPP 的修改版本。新版本 DIALOG3.EXE 与介绍过的 DIALOG2.EXE 非常类似,只是如下部分有所不同:

- 对话框为无模式;
- 对话框有一个额外 Send 下压按钮;
- 主窗口有名为 Find 和 Replace 的两个编辑框。

当单击 Send 按钮时，对话框把 Find 和 Replace 编辑框的文本分别复制到主窗口中的 Find 和 Replace 编辑框中。

当选中 Dialog 菜单项时，应用程序弹出一个模式对话框。对话框中编辑控制 Find 和 Replace 分别出现“ DOS ”和“ Windows ”字符串、Global 圆按钮、Case Sensitive 选中框和 Selected 选中框都初始化为选中状态。键入新文本到一个或两个编辑框，并选 OK 下压按钮。这将导至对话框关闭，对话框的文本将在窗口的编辑框中再现。重新调用对话框，检查是否保持上一次输入的值。现在在两个编辑框中键入新文本，并单击 Send 按钮。可以看见这些文本在窗口编辑框中再现。

清单 20.7 中列出了头文件 DIALOG3.H 的源代码；清单 20.8 为资源描述文件 DIALOG3.RC。清单 20.9 为程序文件 DIALOG3.CPP 的源代码：

键　入　　清单 20.7　头文件 DIALOG3.H 的源代码

```
1: # define CM- EXIT (WM- USER+ 100)
2: # define CM- DIALOG (WM- USER+ 101)
3: # define ID- FIND- TXT 201
4: # define ID- FIND- EDIT 202
5: # define ID- REPLACE- TXT 203
6: # define ID- REPLACE- EDIT 204
7: # define ID- SCOPE- GRP 205
8: # define ID- GLOBAL- RBT 206
9: # define ID- SELTEXT- RBT 207
10: # define ID- CASE- CHK 208
11: # define ID- WHOLEWORD- CHK 209
12: # define ID- FIND- BOX 301
13: # define ID- REPLACE- BOX 302
14: # define ID- SEND- BTN 303
```

键　入　　清单 20.8　资源文件 DIALOG3.RC 的描述

```
1: # include < windows.h>
2: # include < afxres.h>
3: # include "dialog3.h"
4:
5: BUTTONS ACCELERATORS
6: BEGIN
7:    "o", IDOK, ALT
8:    "c", IDCANCEL, ALT
9:    "s", ID- SEND- BTN, ALT
10: END
11:
12: SEARCH DIALOG DISCARDABLE LOADONCALL PURE MOVEABLE 10, 10, 200, 150
13:
14: STYLE WS- POPUP ©¦WS- CLIPSIBLINGS ©¦WS- CAPTION ©¦WS- SYSMENU ©¦
```

```
15:          WS- DLGFRAME ©¦WS- VISIBLE
16:  CAPTION "Controls Demo"
17:  BEGIN
18:    CONTROL "Find", ID- FIND- TXT, "STATIC", WS- CHILD ©¦WS- VISIBLE ©¦
19:    SS- LEFT, 20, 10, 100, 15
20:    CONTROL "", ID- FIND- EDIT, "EDIT", WS- CHILD ©¦WS- VISIBLE ©¦
21:    WS- BORDER ©¦WS- TABSTOP, 20, 25, 100, 15
22:    CONTROL "Replace", ID- REPLACE- TXT, "STATIC", WS- CHILD ©¦
23:    WS- VISIBLE ©¦SS- LEFT, 20, 45, 100, 15
24:    CONTROL "", ID- REPLACE- EDIT, "EDIT", WS- CHILD ©¦WS- VISIBLE ©¦
25:    WS- BORDER ©¦WS- TABSTOP, 20, 60, 100, 15
26:    CONTROL "Scope", ID- SCOPE- GRP, "BUTTON", WS- CHILD ©¦
27:    WS- VISIBLE ©¦WS- GROUP ©¦BS- GROUPBOX, 20, 80, 90, 50
28:    CONTROL "Global", ID- GLOBAL- RBT, "BUTTON", WS- CHILD ©¦
29:    WS- VISIBLE ©¦WS- TABSTOP ©¦BS- AUTORADIOBUTTON, 30, 90, 50, 15
30:    CONTROL "Selected Text", ID- SELTEXT- RBT, "BUTTON", WS- CHILD ©¦
31:    WS- VISIBLE ©¦WS- TABSTOP ©¦BS- AUTORADIOBUTTON, 30, 105, 60, 15
32:    CONTROL "Case Sensitive", ID- CASE- CHK, "BUTTON", WS- CHILD ©¦
33:    WS- VISIBLE ©¦WS- TABSTOP ©¦BS- AUTOCHECKBOX, 20, 130, 80, 15
34:    CONTROL "Whole Word", ID- WHOLEWORD- CHK, "BUTTON", WS- CHILD ©¦
35:    WS- VISIBLE ©¦WS- TABSTOP ©¦BS- AUTOCHECKBOX, 100, 130, 80, 15
36:    CONTROL, "&OK", IDOK, "BUTTON", WS- CHILD ©¦WS- VISIBLE ©¦
37:    WS- TABSTOP ©¦BS- DEFPUSHBUTTON, 120, 90, 30, 20
38:    CONTROL "&Cancel", IDCANCEL, "BUTTON", WS- CHILD ©¦WS- VISIBLE ©¦
39:    WS- TABSTOP ©¦BS- PUSHBUTTON, 160, 90, 30, 20
40:    CONTROL "&Send", ID- SEND- BTN, "BUTTON", WS- CHILD ©¦WS- VISIBLE ©¦
41:    WS- TABSTOP ©¦BS- PUSHBUTTON 160, 60, 30, 20
42:  END
43:
44:  MAINMENU MENU LOADONCALL MOVEABLE PURE DISCARDABLE
45:  BEGIN
46:    MENUITEM "E&xit", CM- EXIT
47:    MENUITEM "&Dialog", CM- DIALOG
48:  END
```

键　入　　**清单 20.9　程序文件 DIALOG3.CPP 的源代码**

```
1:  # include < string.h>
2:  # include < afxwin.h>
3:  # include "dialog3.h"
4:
5:  // declare constants for sizing and dimensioning the
6:  // application window's controls
7:  const Wtxt= 200;
8:  const Htxt= 20;
9:  const TxtVertSpacing= 10;
10:  const Wbox= 200;
11:  const Hbox= 30;
12:  const BoxHorzSpacing= 40;
```

```
13: // declare other constants
14: const MaxEditLen= 30;
15: const BF_CHECKED= 1;
16: const BF_UNCHECKED= 0;
17:
18: // declare the application buffer
19: struct TAppTransferBuf {
20:   char FindBoxBuff[MaxEditLen+ 1];
21:   char ReplaceBoxBuff[MaxEditLen+ 1];
22:   int GlobalRbtBuff;
23:   int SelTextRbtBuff;
24:   int CaseChkBuff;
25:   int WholeWordChkBuff;
26: };
27:
28: class CWindowApp: public CWinApp
29: {
30: public:
31:   virtual BOOL InitInstance();
32: };
33:
34: // expand the functionality of CDialog by
35: // deriving class CAppDialog
36: class CAppDialog : public CDialog
37: {
38: public:
39:
40:   CAppDialog(const char FAR* lpTemplateName,
41:   CWnd* pParentWnd);
42:   // handle pressing the OK button
43:   afx_msg void OnOK();
44:   afx_msg void CMOK() {OnOK(); }
45:   // handle pressing the Cancel button
46:   afx_msg void OnCancel();
47:   afx_msg void CMCancel() { OnCancel(); }
48:   // handle sending data without closing the modal dialog box
49:   afx_msg void HandleSendBtn();
50:   afx_msg void CMSendBtn() { HandleSendBtn();}
51:
52: protected:
53:
54: // pointer to data buffer in parent window
55: TAppTransferBuf* pAppBuffer;
56:
57: // handle initializing the dialog box
58: virtual BOOL OnInitDialog();
59: // transfer data from the controls to the buffer
60: void TransferData();
61:
62: // declare message map macro
63: DECLARE_MESSAGE_MAP();
64: };
```

```
65:
66: // expand the functionality of CFrameWnd by deriving
67: // class CAppWindow
68: class CAppWindow: public CFrameWnd
69: {
70: public:
71:
72:   // declare data transfer buffer
73:   TAppTransferBuf AppBuffer;
74:   // flag to indicate whether or not the
75:   // modeless dialog box is active
76:   BOOL bDialogActive;
77:
78:   CAppWindow();
79:   ~CAppWindow();
80:   // handle the dialog command
81:   afx_msg void CMDialog();
82:   // handle a message sent by the Send button of
83:   // the dialog box
84:   afx_msg void HandleSendBtn();
85:   // handle exiting the application
86:   afx_msg void OnExit()
87:   { SendMessage(WM_CLOSE); }
88:
89: protected:
90:
91:   // declare the pointers to the window's controls
92:   // and the modeless dialog box
93:   CStatic* FindTxt;
94:   CEdit* FindBox;
95:   CStatic* ReplaceTxt;
96:   CEdit* ReplaceBox;
97:   CAppDialog* pDlg;
98:
99:   // handle creating the controls
100:   afx_msg int OnCreate(LPCREATESTRUCT lpCS);
101:   // handle closing the window
102:   afx_msg void OnClose();
103:   // return a reference to member r based on individual
104:   // coordinates and dimenions
105:   void makeRect(int nX, int nY, int nW, int nH, CRect& r);
106:
107:   // declare message map macro
108:   DECLARE_MESSAGE_MAP();
109:
110: };
111:
112: CAppDialog::CAppDialog(const char FAR* lpTemplateName,
113:                        CWnd* pParentWnd)
114: {
115:   Create(lpTemplateName, pParentWnd);
116: }
```

```
117:
118: BOOL CAppDialog::OnInitDialog()
119: {
120:   CAppWindow*  pW= (CAppWindow* ) (GetParent());
121:   pAppBuffer= &pW- > AppBuffer;
122:   // copy data from the buffer to the dialog box controls
123:   SetDlgItemText(ID_FIND_EDIT,pAppBuffer- > FindBoxBuff);
124:   SetDlgItemText(ID_REPLACE_EDIT,pAppBuffer- > ReplaceBoxBuff);
125:   CButton*  pBtn= (CButton* )(GetDlgItem(ID_GLOBAL_RBT));
126:   pBtn- > SetCheck(pAppBuffer- > GlobalRbtBuff);
127:   pBtn= (CButton* )(GetDlgItem(ID_SELTEXT_RBT));
128:   pBtn- > SetCheck(pAppBuffer- > SelTextRbtBuff);
129:   pBtn= (CButton* ) (GetDlgItem(ID_CASE_CHK));
130:   pBtn- > SetCheck(pAppBuffer- > CaseChkBuff);
131:   pBtn= (CButton* )(GetDlgItem(ID_WHOLEWORD_CHK));
132:   pBtn- > SetCheck(pAppBuffer- > WholeWordChkBuff);
133:   return TRUE;
134: }
135:
136: void CAppDialog::OnOK()
137: {
138:   CAppWindow*  pParent= (CAppWindow* ) (GetParent());
139:   // transfer data to the buffer
140:   TransferData();
141:   // notify parent of transfer
142:   pParent- > SendMessage(WM_COMMAND,ID_SEND_BTN);
143:   // clear active dialog flag
144:   pParent- > bDialogActive= FALSE;
145:   // close the modeless dialog box
146:   DestroyWindow();
147: }
148:
149: void CAppDialog::OnCancel()
150: {
151:   CAppWindow*  pParent= (CAppWindow* ) (GetParent());
152:   // clear active dialog flag
153:   pParent- > bDialogActive= FALSE;
154:   // close the modeless dialog box
155:   DestroyWindow();
156: }
157:
158: void CAppDialog::HandleSendBtn()
159: {
160:   CAppWindow*  pParent= (CAppWindow* ) (GetParent());
161:
162:   // transfer data to the buffer
163:   TransferData();
164:   // notify parent of transfer
165:   pParent- > SendMessage(WM_COMMAND,ID_SEND_BTN);
167: }
168: void CAppDialog::TransferData()
169: {
```

```
170:    // copy data from the dialog box controls to the buffer
171:    GetDlgItemText(ID_FIND_EDIT, pAppBuffer->FindBoxBuff,
172:    MaxEditLen);
173:    GetDlgItemText(ID_REPLACE_EDIT, pAppBuffer->ReplaceBoxBuff,
174:    MaxEditLen);
175:    CButton* pBtn= (CButton* )(GetDlgItem(ID_GLOBAL_RBT));
176:    pAppBuffer->GlobalRbtBuff= pBtn->GetCheck();
177:    pBtn= (CButton* )(GetDlgItem(ID_SELTEXT_RBT));
178:    pAppBuffer->SelTextRbtBuff= pBtn->GetCheck();
179:    pBtn= (CButton* )(GetDlgItem(ID_CASE_CHK));
180:    pAppBuffer->CaseChkBuff= pBtn->GetCheck();
181:    pBtn= (CButton* )(GetDlgItem(ID_WHOLEWORD_CHK));
182:    pAppBuffer->WholeWordChkBuff= pBtn->GetCheck();
183: }
184:
185: CAppWindow::CAppWindow()
186: {
187:    // load the accelerator keys resource
188:    LoadAccelTable("BUTTONS");
189:
190:    // create the window
191:    Create(NULL, "Modeless Dialog Box Data Transfer Tester",
192:          WS_OVERLAPPEDWINDOW, rectDefault, NULL, "MAINMENU");
193:
194:    // fill buffer with 0's
195:    memset(&AppBuffer, 0x0, sizeof(AppBuffer));
196:    strcpy(AppBuffer.FindBoxBuff, "DOS");
197:    strcpy(AppBuffer.ReplaceBoxBuff, "Windows");
198:    AppBuffer.GlobalRbtBuff= BF_CHECKED;
199:    AppBuffer.CaseChkBuff= BF_CHECKED;
200:    AppBuffer.WholeWordChkBuff= BF_CHECKED;
201:    // clear the active dialog flag
202:    bDialogActive= FALSE;
203: };
204:
205: CAppWindow::~CAppWindow()
206: {
207:    delete FindTxt;
208:    delete FindBox;
209:    delete ReplaceTxt;
210:    delete ReplaceBox;
211: }
212:
213: int CAppWindow::OnCreate(LPCREATESTRUCT lpCS)
214: {
215:    int nX0= 30;
216:    int nY0= 20;
217:    int nX= nX0;
218:    int nY= nY0;
219:
220:    CRect r;
221:    DWORD dwStaticStyle= WS_CHILD ©¦WS_VISIBLE ©¦SS_LEFT;
```

```
222:   DWORD dwEditStyle= WS- CHILD ©¦WS- VISIBLE ©¦WS- BORDER ©¦
223:   ES- LEFT ©¦ES- AUTOHSCROLL;
224:   // create Find static text
225:   makeRect(nX, nY, Wtxt, Htxt, r);
226:   FindTxt= new CStatic();
227:   FindTxt- > Create("Find", dwStaticStyle, r, this);
228:   // create Find edit control
229:   nY + = Htxt + TxtVertSpacing;
230:   makeRect(nX, nY, Wbox, Hbox, r);
231:   FindBox= new CEdit();
232:   FindBox- > Create(dwEditStyle, r, this, ID- FIND- BOX);
233:   // create Replace static text
234:   nX + = Wbox + BoxHorzSpacing;
235:   nY= nY0;
236:   makeRect(nX, nY, Wtxt, Htxt, r);
237:   ReplaceTxt = new CStatic();
238:   ReplaceTxt- > Create("Replace", dwStaticStyle, r, this);
239:   // create Replace edit control
240:   nY + = Htxt + TxtVertSpacing;
241:   makeRect(nX, nY, Wbox, Hbox, r);
242:   ReplaceBox= new CEdit();
243:   ReplaceBox- > Create(dwEditSylte, r, this, ID- REPLACE- BOX);
244:   return CFrameWnd::OnCreate(lpCS);
245: }
246:
247: void CAppWindow::CMDialog()
248: {
249:   if (bDialogActive) return;
250:   pDlg= new CAppDialog("SEARCH", this);
251:   bDialogActive= TRUE;
252:   pDlg- > SendMessage(WM- INITDIALOG);
253: }
254:
255: void CAppWindow::HandleSendBtn()
256: {
257:   FindBox- > SetWindowText(AppBuffer.FindBoxBuff);
258:   ReplaceBox- > SetWindowText(AppBuffer.ReplaceBoxBuff);
259: }
260:
261: void CAppWindow::OnClose()
262: {
263:   if (MessageBox("Want to close this application",
264:               "Query", MB- YESNO ©¦WS- ICONQUESTION)= = IDYES)
265:     DestroyWindow();
266: }
267:
268: void CAppWindow::makeRect(int nX, int nY, int nW, int nH,
269:                           CRect& r)
270: {
271:   r.top= nY;
272:   r.left= nX;
273:   r.bottom= nY+ nH;
```

```
274:    r.right= nX+ nW;
275: }
276:
277: BEGIN_MESSAGE_MAP(CAppDialog, CDialog)
278:                 ON_BN_CLICKED(IDOK, OnOK)
279:    ON_COMMAND(IDOK, CMOK)
280:    ON_BN_CLICKED(IDCANCEL, OnCancel)
281:    ON_COMMAND(IDCANCEL, CMCancel)
282:    ON_BN_CLICKED(ID_SEND_BTN, HandleSendBtn)
283:    ON_COMMAND(ID_SEND_BTN, CMSendBtn)
284: END_MESSAGE_MAP()
285:
286: BEGIN_MESSAGE_MAP(CAppWindow, CFrameWnd)
287:    ON_COMMAND(CM_DIALOG, CMDialog)
288:    ON_COMMAND(ID_SEND_BTN, HandleSendBtn)
289:    ON_COMMAND(CM_EXIT, OnExit)
290:    ON_WM_CREATE()
291:    ON_WM_CLOSE()
292: END_MESSAGE_MAP()
293:
294: // Construct the CWindowApp's m_pMainWnd data member
295: BOOL CWindowApp::InitInstance()
296: {
297:    m_pMainWnd= new CAppWindow();
298:    m_pMainWnd->ShowWindow(m_nCmdShow);
299:    m_pMainWnd->UpdateWindow();
300:    return TRUE;
301: }
302:
303: // application's constructor initializes and runs the app
304: CWindowApp WindowApp;
```

图 20.4 为应用程序 DIALOG3.EXE 的一个运行范例。

分　析

清单 20.8 中列出了 DIALOG3.RC 资源描述文件。资源文件定义了加速键、菜单和无模式对话框资源。

清单 20.9 为程序文件 DIALOG3.CPP 的源代码。清单中声明了一组用于在窗口应用程序中指定控制的尺寸和间距的常量。清单中还包括数据传送缓冲区类型——和程序 DIALOG2.EXE 中一样——的声明。

应用程序声明了三个类:应用程序类、窗口类和对话框类。CAppDialog 构造函数只简单调用继承而来的 Create 函数建立一个对话框。类 CAppDialog 声明了处理每个下压按钮的成员函数。对话框类声明了用于把数据从对话框控制传送到缓冲区的成员函数 TransferData,此函数被成员函数 OnOK 和 HandleSendBtn 调用。

第 136 到 147 行定义了虚成员函数 OnOK,以响应 OK 按钮发出的 IDOK 通知消息,函数执行以下任务:

图 20.4　应用程序 DIALOG3.EXE 的一个运行范例。

· 调用 TransferData 成员函数，把控制数据传送到缓冲区；
· 使用 SendMessage 函数发送一条 ID_SEND_BTN 消息给父窗口；
· 设置父窗口数据成员 bDialogActive 为 FALSE；
· 调用 DestroyWindow 函数关闭对话框。

第 149 到 156 行定义了成员函数 OnCancel，以处理 Cancel 按钮所发出的 IDCANCEL 通知消息。本函数执行如下的两个任务：

· 设置父窗口数据成员 bDialogActive 为 FALSE；
· 调用 DestroyWindow 函数关闭对话框。

第 255 到 259 行定义了成员函数 HandleSendBtn，以处理 Send 按钮发出的 ID_SEND_BTN 通知消息。它执行如下的任务：

· 调用 TransferData 成员函数，把控制数据传送到缓冲区中；
· 调用 SendMessage 函数发送 ID_SEND_BTN 消息给父窗口。

成员函数 CMSendBtn 处理加速键 Alt-S 产生的 ID_SEND_BTN 命令消息，此函数只简单地调用 HandleSendBtn 成员函数。

应用程序窗口类 CAppWindow 声明了两个公有数据成员、五个保护数据成员、一个构造函数、一个析构函数和六个成员函数。布尔型数据成员 bDialogActive 表明对话框实例是否存在，使用这个标志可以确保只有一个无模式对话框实例。数据成员 AppBuffer 是数据传送缓冲区。窗口类声明了五个保护指针。FindBox 和 ReplaceBox 指针用于存取编辑框实例；pDlg 为指向对话框实例的指针。

CAppWindow 构造函数执行如下任务:

· 装入菜单资源;
· 调用 Create 函数建立应用程序窗口;
· 用零填充缓冲区, 对其初始化;
· 赋值给编辑框缓冲区、选中框缓冲区和 Global 圆按钮缓冲区;
· 给 bDialogActive 数据成员赋值为 FALSE。

第 213 到 245 行定义了成员函数 OnCreate, 实质上建立附属于应用程序窗口的控制。此函数执行如下的任务:

· 建立标名为 Find 的编辑框的静态文本实例;
· 建立 Find 编辑框;
· 建立标名为 Replace 的编辑框的静态文本实例;
· 建立 Replace 编辑框;
· 返回函数值。

应用程序窗口类声明了成员函数 CMDialog, 用于建立一个无模式对话框, 此函数执行如下的任务:

· 如果 bDialogActive 成员为 TRUE 时, 退出;
· 建立由 pDlg 指针存取的 CAppDialog 实例;
· bDialogActive 数据成员赋值为 TRUE
· 发送一条 WM_INITDIALOG 消息, 在屏幕上产生无模式对话框。(本任务中使用了 pDlg 指针使对话框接收想要的消息)。

第 255 到 259 行定义了成员函数 HandleSendBtn, 以响应对话框的 Send 按钮发出的消息并拷贝 FindBoxBuff 和 ReplaceBoxBuff 缓冲区成员到窗口的 Find 和 Replace 编辑控制中。

小　　结

今天的课程介绍了功能强大的输入工具—— 对话框。我们学习了以下的课题:

· 构造一个类 CDialog 实例, 建立模式或无模式对话框。
· 模式对话框用 DoModal 成员函数执行。
· 传送控制数据的基本步骤包括声明数据传送缓冲区类型、声明缓冲区、按相应缓冲区的顺序建立控制以及建立缓冲区链。
· 要进行数据传送, 首先是定义一个传送缓冲区类型。既可以传送模式对话框数据、也可以传送无模式对话框的数据。

问　与　答

问: MFC 支持专门对话框吗?

答: 是, MFC 库有一个类集来实现对话框。这些对话框可用于选择文件、选择颜色、选择字体、打印和搜索/替换文本。这些对话框的模型类都是 CDialog 类的子孙类。

问: MFC 1.0 版提供的对话框类有所不同吗?

答: 是的, 1.0 版提供的 CModalDialog 对话框是类 CDialog 的子孙类, 2.0 版使用 CDialog 支持两种类型的对话框, 编译器能识别出类 CModalDialog, 将其作为 CDialog 处理。

问: 可以紧跟着模式对话框之后生成一个无模式对话框集吗?

答: 可以。有趣的是, 模式对话框允许选择在它之前建立的无模式对话框, 但是, 在关闭模式对话框之前, 不能另外建立无模式对话框。

问: 无模式对话框必须有数据传送缓冲区吗?

答: 不总是需要, 可以有弹出多个无模式对话框的应用程序, 让它们互相直接通信, 而不需要数据传送缓冲区。

问: 用工具(如 App Studio)可以方便地写自己的. RC 资源文件吗?

答: 可以。. RC 资源文件描述对话框控制与其他资源的严格尺寸、风格和其他属性。

问: . RC 资源文件被编译成什么文件?

答: 资源编译器 RC. EXE 把. RC 文件编译成. RES 文件。

作　　业

作业部分所提供的测验题有助于巩固理解所学内容, 通过练习可以积累运用所学知识的经验。要尽量弄懂测验和练习的答案后再继续下一天的课程。附录 B 的‘答案’中提供了解答。

测　验

1. 对还是错?
 在应用程序的建立之前或期间, 必须把所有的. RC 文件编译成. RES。
2. 对还是错?
 对话框的 OK 和 Cancel 按钮是可选项。
3. 对还是错?
 可以创造一个带名为 Yes 和 No 按钮的对话框。
4. 对还是错?
 Windows 不允许嵌套对话框。
5. 对还是错?
 对话框必须总有一个非对话的父窗口。

练　习

1. 建立命令行计算器应用程序(COCA)的一个版本,使用对话框作为独立窗口。
2. 使用 Studio App 为程序 DIALOG2.CPP 建立对话框资源。
3. 使用 Studio App 为程序 DIALOG3.CPP 建立对话框资源。

第21天

MDI 窗 口

回想一下，许多 Windows 应用程序，如 Windows Progam Manager，Windows File Manager 和许多 Windows 文本编辑器，它们都实现了一种特殊的窗口界面——多文档界面(MDI)。MDI 标准是 IBM 提出的 CUA(Common User Access，即公共用户访问)标准的一部分。每一个服从 MDI 的应用程序，允许打开子窗口以用于对特定文件的操作，如编辑文本、管理数据库和处理电子数据表格。今天这一课，我们将学习以下有关管理 MDI 窗口和对象的内容：

- MDI 应用程序的基本特性和组成部分
- 建立 MDI 应用程序
- CMDIFrameWnd 类
- 建立 MDI 主窗口
- CMDIChildWnd 类
- 建立 MDI 子窗口

21.1 MDI 应用程序的特性及其组成

一个 MDI 应用程序由如下的对象组成：

- MDI 主窗口是类 CMDIFrameWnd 的一个实例，每个 MDI 应用程序有一个 MDI 主框窗口。
- MDI 客户窗口由 CreateClient 成员函数建立，每个 MDI 应用程序存一个 MDI 客户窗口。
- MDI 子窗口为 CMDIChildWnd 或其子孙类的一个实例，一个 MDI 应用程序动态地建立和删除 MDI 子窗口的多个实例，在任一指定时间(此时，至少有一个 MDI 子窗口)，只有一个活动的 MDI 子窗口。

新 词

MDI 主窗口 (MDI frame window) MDI 客户窗口 (MDI client window) MDI 子窗口 (MDI child window)

可视的 MDI 主窗口包括所有其它的 MDI 对象，不可视的 MDI 客户窗口对动态创建和删除的 MDI 子窗口进行底层管理。动态和可视的 MDI 子窗口只能在 MDI 主窗口定义的范围内定位、移动、改变大小、极大化和最小化。

当一个 MDI 子窗口极大化时，它占领 MDI 主窗口限定的区域。极小化一个 MDI 子

窗口时,窗口的图标会出现在 MDI 主窗口的底部。

> **注　意**　MDI 主窗口有一个菜单,使用它可以操作 MDI 子窗口及其内容。MDI 子窗口不能有菜单,但可含有控制。MFC 库支持以下特性:当选择一个不同的 MDI 子窗口时,MDI 主窗口的附属菜单也会改变。这种特性使各个 MDI 子窗口可以有不同的数据和用途。

21.2　建立一个 MDI 应用程序

通过上一节我们已经知道,一个 MDI 应用程序的基本组成部分是类 CMDIFrameWnd 和 CMDIChildWnd,CMDIFrameWnd 类支持如下的任务:

- 建立和处理 MDI 客户窗口
- 建立和处理 MDI 子窗口
- 管理菜单选项

21.3　CMDIFrameWnd 类

MFC 提供了 CMDIFrameWnd 类,它是 CFrameWnd 子孙类,用于实现 MDI 应用程序的 MDI 主窗口。

CMDIFrameWnd 类声明了一个构造函数、一个 Create 函数和一些用于操作 MDI 子窗口的成员函数。

Create 函数是继承而来的,它与类构造函数一起建立类 CMDIFrameWnd 实例。它的各个参数值定义了 Windows 类(一个 WINDCLASS 结构),MDI 窗口标题,窗口风格,窗口的风格和位置(由 CRect 参数指定),指向父窗口的指针和相关的菜单资源名。当 MDI 主窗口是最上一级窗口时,父窗口指针取值为 NULL。成员函数 CreateClient 用来建立不可见 MDI 客户窗口,此函数有两个参数:一个是 LPCREATSTRUCT,另一个是 CMenu * 菜单指针。

CMDI FrameWnd 的成员函数支持对 MDI 子窗口的若干操作。这些操作包括:

- 用成员函数 MDIActive 激活 MDI 子窗口
- 用成员函数 MDIGetActive 获得指向当前活动的 MDI 子窗口的指针
- 使用成员函数 MDIMaximize 和 MDIRestore 分别极大化和恢复一个 MDI 子窗口
- 使用 MDICascade 和 MDITile 成员函数分别使 MDI 子窗口成为层叠式和平铺式。
- 选择下一个 MDI 子窗口(在当前活动 MDI 子窗口的下面)。

21.4　建立 MDI 主窗口

建立组成一个 MFC 应用程序的对象的常用方法是先建立应用程序实例随后建立它

的主窗口实例。在 MDI 应用程序中，应用程序主窗口通常是类 CMDIFrameWnd 的一个子孙，应用程序类的 InitInstance 成员函数用来建立此窗口。MDI 主窗口类的构造函数与 Create 函数一起建立 MDI 框架和 MDI 客户窗口，也可以建立一个初始化的或缺省的 MDI 子窗口。例如，一些 Windows 文本编辑器打开上次编辑的文件(如果可能的话)，而另一些编辑器则显示一个空的 MDI 子窗口。

21.5　CMDIChildWnd 类

MFC 提供了 CMDIChildWnd 类，它是 CFrameWnd 的子孙类，以实现 MDI 子窗口。CMDIChildWnd 类定义了一个构造函数、一个 Create 函数、一组成员函数和一个数据成员。成员函数允许对 MDI 子窗口进行删除、激活、极大化、恢复及获得指向父窗口的指针。

21.6　建立 MDI 子窗口

建立 MDI 子窗口与本书先前介绍的建立一个应用程序窗口基本类似，不同之处有：

- MDI 子窗口是 CMDIChildWnd 或其子孙类的实例。
- MDI 子窗口不能有自己的菜单，MDI 主窗口的菜单用于操纵当前活动的 MDI 子窗口或所有的 MDI 子窗口。
- MDI 子窗口可以有控制——虽不常见但是允许。

21.7　管理 MDI 消息

消息循环先直接把命令消息送给活动的 MDI 子窗口，以允许其进行响应，如子窗口没有响应，随后把消息送到父窗口，即 MDI 主窗口。当然，活动的 MDI 子窗口响应其控制发出的通知消息的方式和一般窗口或对话框是一样的。

21.8　一个简单的文本查阅器

现在，让我们看一个简单的 MDI 应用程序。MDI 应用程序经常被用作文本查阅器和文本编辑器。本应用程序模拟了一个简单的文本查阅器——应用程序实际上显示的是随机文本，而不是能从文件里获得的文本。这种方法，可使程序简单，并让你把注意力集中在各种 MDI 对象的使用上。此 MDI 应用程序有一个包含 Exit 和 MDI 子菜单项的简单菜单。

现在编译并运行此程序，尝试着创建 MDI 子窗口。注意，奇数号的 MDI 子窗口是静态的，而偶数号的 MDI 子窗口中的正文则可以编辑。此特性演示如何建立一个文本查阅器和文本编辑器。(没有装入和保存选择，以使例子简短)。尝试对子窗口的极大化和极小化，以及使它成为层叠式或平铺式，也可以测试关闭一个 MDI 子窗口和关闭所有 MDI 子窗口的情况。

清单 21.1 中列出了头文件 MDIWIN1.H 的源代码；清单 21.2 为资源描述文件

MDIWIN1. RC 的代码; 清单 21. 3 为程序 MDIWIN1. CPP 的源代码。

图 21. 1 为程序 MDIWIN1. EXE 的一个运行范例。

图 21. 1　程序 MDIWIN1. EXE 的运行范例

键　入　　清单 21. 1　头文件 MDIWIN1. H 的源代码

```
1: # define CM_EXIT (WM_USER+ 100)
2: # define CM_COUNTCHILDREN (WM_USER+ 101)
3: # define CM_CREATECHILD (WM_USER+ 102)
4: # define CM_CASCADECHILDREN (WM_USER+ 103)
5: # define CM_TILECHILDREN (WM_USER+ 104)
6: # define CM_ARRANGEICONS (WM_USER+ 105)
7: # define CM_CLOSECHILDREN (WM_USER+ 106)
8: # define WM_CHILDDESTROY (WM_USER+ 200)
9:
10: # define ID_TEXT_EDIT 100
```

键　入　　清单 21. 2　资源文件 MDIWIN1. RC 的源代码

```
1: # include < windows. h>
2: # include < afxres. h>
3: # include "mdiwin1. h"
4:
5: MAINMENU MENU LOADONCALL MOVEABLE PURE DISCARDABLE
6: BEGIN
```

```
7: MENUITEM "E&xit", CM_EXIT
8: POPUP "&MDI Children"
9: BEGIN
10:    MENUITEM "C&reate", CM_CREATECHILD
11:    MENUITEM "&Cascade", CM_CASCADECHILDREN
12:    MENUITEM "&Tile", CM_TILECHILDREN
13:    MENUITEM "Arrange &Icons", CM_ARRANGEICONS
14:    MENUITEM "C&lose All", CM_CLOSECHILDREN
15:    MENUITEM "C&ount Children", CM_COUNTCHILDREN
16:    END
17: END
```

键　入　　**清单 21.3　程序文件 MDIWIN1.CPP 的源代码**

```
1: #include <std1ib.h>
2: #include <stdio.h>
3: #inculde <string.h>
4: #include <afxwin.h>
5: #include "mdiwin1.h"
6:
7: const char *  MenuRezName= "MAINMENU";
8: const Hct1= 200;
9: const Wct1= 400;
10: const MaxWords= 200;
11: const WordsPerLine= 10;
12: const NumWords= 10;
13: char *  Words[NumWords]= {"The" "man", "ate", "the",
14: "food", "in", "hurry", "girl",
15: "cake", "bread"};
16: // declare a global random- number generating function
17: int random(int maxVal)
18: { return rand() % maxVal;}
19:
20: // declare global function to calculate the size and
21: // location of a control
22: void makeRect(int nX, int nY, int nW, int nH, CRect& r)
23: {
24:    r.top= nY;
25:    r.left= nX;
26:    r.bottom= nY+ nH;
27:    r.right= nX+ nW;
28: }
29:
30: // Define an application class derived from CWinApp
31: class CWindowApp:  public CWinApp
32: {
33:    public:
34:
35:     virtual BOOL InitInstance();
```

```
36: };
37:
38: class CAppMDIChild：  public CMDIChildWnd
39: {
40: public:
41:   int nChildNum;
42:   // pointer to the edit box control
43:   CEdit*  TextBox;
44:   CStatic*  TextTxt;
45:
46: CAppMDIChild();
47: ～CAppMDIChild();
48: BOOL Create(LPSTR szTitle,int nChildNum);
49:
50: protected:
51: // handle closing the MDI child window
52: afx_msg void OnClose();
53: // handle destroying an MDI child window
54: afx_msg void OnDestroy();
55: afx_msg int OnCreate(LPCREATESTRUCT lpCS);
56:
57: // declare message map macro
58: DECLARE_MESSSAGE_MAP()
59: };
60:
61: class CAppMDIFrame：  public CMDIFrameWnd
62: {
63: // DECLARE_DYNCREATE(CAppMDIFrame)
64: public:
65: // flag to quickly close all MDI children windows
66: BOOL bExpressClose;
67: int nLastMDIChild;
68:
69: CApp MDIFrame();
70: ～CAppMDIFrame() {delete pMenu; }
71: // create a new client window area
72: afx_msg int OnCreate (LPCREATESTRUCT lpCS);
73: // create an MDI child window
74: afx_msg void CMCreateChild();
75: // cascade MDI children
76: afx_msg void CMCascadeChildren() { MDICascade(); }
77: // tile MDI childern
78: afx_msg void CMTileChildern() { MDITile(); }
79: // arrange MDI childern icons
80: afx_msg void CMArrangeChildern();
81: // close all MDIchildern
82: afx_msg void CMCloseChildern();
83: // get the number of MDI childern
84: int GetChildCount() { return nNumMDIChildern; }
85: // handle the command for counting the MDI childern
86: afx_msg void CMCountChildren();
87: // handle the child destroy message
```

```
88: afx_msg LONG OnChildDestroy(UINT wParam,LONG lParam);
89: // handle exiting the application
90: afx_msg void OnExit()
91: { SendMessage(WM_CLOSE); }
92:
93: protected:
94:   CMenu*  pMenu;
95:   int nNumMDIChildren;
96:   // handle closing the MDI frame window
97:   virtual void OnClose();
98:
99:   // declare message map macro
100:    DECLARE_MESSAGE_MAP();
101: };
102:
103: CAppMDIChild::CAppMDIChild()
104: {
105:   TextBox= NULL;
106:   TextTxt= NULL;
107: }
108:
109: CAppMDIChild::~CAppMDIChild()
110: {
111: if (TextBox)
112:    delete TextBox;
113: else
114:    delete TextTxt;
115: }
116:
117: BOOL CAppMDIChild::Create(LPSTR szTitle,int nChildNumber)
118: {
119:   nChildNum= nChildNumber;
120:   return CMDIChildWnd::Create(NULL,szTitle);
121: }
122:
123: CAppMDIChild::OnCreate(LPCREATESTRUCT lpCS)
124: {
125:   CString s;
126:   CRect r;
127:   DWORD dwStaticStyle= WS_CHILD©¦WS_VISIBLE©¦SS_LEFT;
128:   DWORD dwEditStyle= WS_CHILD©¦WS_VISIBLE©¦ES_LEFT©¦
129:   ES_MULTILINE©¦ES_AUTOHSCROLL©¦
130:   ES_AUTOVSCROLL;
131:
132:   // randomize seed for the random-number generator
133:   srand(unsigned)time(NULL));
134:   // build the list of random words
135:   for (int i= 0;i < MaxWords;i++) {
136:    if (i > 0 && i % WordsPerLine= = 0)
137:       s + = "\r\n";
138:    s + = Words[random(NumWords)];
139:    }
```

```
140:    // obtain the size and location of the control to create
141:    makeRect(10, 10, Wct1, Hct1, r);
142:    // create a static text object in the child window if the
143:    // nChildNum variable stores an odd number. Otherwise,
144:    // create an edit box control
145:    if (nChildNum % 2= = 0) {
146:      // create the edit box
147:      TextBox= new CEdit();
148:      TextBox- > Create(dwEditStyle, r, this, ID- TEXT- EDIT);
149:      TextBox- > SetWindowText((const char* ) s);
150:    }
151:    else {
152:    // create static text
153:    TextTxt= new CStatic();
154:    TextTxt- > Create((const char* )s, dwStaticStyle, r, this);
155:    }
156: return 0;
157: }
158:
159: void CAppMDIChild::OnClose()
160: {
161:    CAppMDIFrame* pParent= (CAppMDIFrame* )(GetParentFrame());
162:
163:    // return TRUE if the bExpressClose member of the parent window set
164:    // is the express- close flag of the parent window set?
165:    if (pParent- > bExpressClose= = TRUE)
166:     return;
167:    else
168:     // prompt the user and return the prompt result
169:    if (MessageBox("Close this MDI window?",
170:                "Query", MB- YESNO|MB- ICONQUESTION)= = IDYES) {
171:     MDIDestroy();
172:    }
173: }
174:
175: void CAppMDIChild::OnDestroy()
176: {
177:    CAppMDIFrame* pParent= (CAppMDIFrame* )(GetParentFrame());
178:    pParent- > SendMessage(WM- CHILDDESTROY, (UINT)m- hWnd, 0);
179: }
180:
181: CAppMDIFrame::CAppMDIFrame()
182: {
183:    Create(NULL, "MDI Text Viewer",
184:    WS- OVERLAPPEDWINDOW|WS- MAXIMIZE, rectDefault,
185:    NULL, MenuRezName);
186:    // clear the express- close flag
187:    bExpressClose= FALSE;
188:    // initialize the number of MDI childern
189:    nLastMDIChild= 0;
190:    nNumMDIChildren= 0;
191: }
```

```
192:
193: int CAppMDIFrame::OnCreate(LPCREATESTRUCT lpCS)
194: {
195:   pMenu= new CMenu();
196:   pMenu- > LoadMenu(MenuRezName);
197:   CreateClient(lpCS, pMenu- > GetSubMenu(0));
198:   return 0;
199: }
200:
201: void CAppMDIFrame::CMCreateChild()
202: {
203:   char s[81];
204:   CRect r;
205:   CAppMDIChild* pChild= new CAppMDIChild();
206:   makeRect(10 * nNumMDIChildren, 10 * nNumMDIChildren,
207:         Wct1, Hct1, r);
208:   nLastMDIChild+ + ;
209:   sprintf(s, "Child # %d", nLastMDIChild);
210:   if (! pChild- > Create(s, nLastMDIChild)) {
211:    delete pChild;
212:   nLastMDIChild- - ;// decrement highest child index
213:   return;
214: }
215: nNumMDIChildren+ + ;
216: // show the new MDI child window
217: pChild- > ShowWindow(WS- SHOW);
218: }
219:
220: void CAppMDIFrame::CMCloseChildren()
221: {
222:   CAppMDICHild* pChild= (CAppMDIChild* ) MDIGetActive();
223:
224:   // set the bExpressClose flag
225:   bExpressClose= TRUE;
226:   while (pChild) {
227:   // close the active MDI child
228:   pChild- > MDIDestroy();
229:   // get the next MDI child
230:   pChild= (CAppMDIChild* ) MDIGetActive();
231: }
232: // clear the bExpressClose flag
233: bExpressClose= FALSE;
234: }
235:
236: // display a message box which shows the number of children
237: void CAppMDIFrame::CMCountChildren()
238: {
239: char msgStr[81];
240:
241: sprintf(msgStr, "There are %d MDI children", nNumMDIChildren);
242: MessageBox(msgStr, "Information", MB- OK©MB- ICONINFORMATION);
243: }
```

```
244:
245: LONG CAppMDIFrame::OnChildDestroy(UINT wParam,LONG lParam)
246: {
247:   nNumMDIChildren--;
248:   // reset nLastMDIChild to 0,if there are no MDI children
249:   nLastMDIChild=(nNumMDIChildren==0)?0:nLastMDIChild;
250:   return 0;
251: }
252:
253: void CAppMDIFrame::OnClose()
254: {
255:   if(MessageBox("Want to close this application",
256:              "Query",MB_YESNO|MB_ICONQUESTION)==IDYES)
257:    DestroyWindow();
258: }
259:
260: BEGIN_MESSAGE_MAP(CAppMDIChild,CMDIChildWnd)
261:   ON_WM_CREATE()
262:   ON_WM_CLOSE()
263:   ON_WM_DESTROY()
264: END_MESSAGE_MAP()
265:
266: BEGIN_MESSAGE_MAP(CAppMDIFrame,CMDIFrameWnd)
267:   ON_WM_CREATE()
268:   ON_COMMAND(CM_CREATECHILD,CMCreateChild)
269:   ON_COMMAND(CM_CASCADECHILDREN,CMTileChildren)
270:   ON_COMMAND(CM_TILECHILDREN,CMTileChildren)
271:   ON_COMMAND(CM_ARRANGEICONS,CMArrageIcons)
272:   ON_COMMAND(CM_CLOSECHILDREN,CMCloseChildren)
273:   ON_COMMAND(CM_COUNTCHILDREN,CMCountChildren)
274:   ON_COMMAND(WM_CHILDDESTROY,OnChildDestroy)
275:   ON_COMMAND(CM_EXIT,OnExit)
276:   ON_WM_CLOSE()
277: END_MESSAGE_MAP()
278:
279: // construct the CWindowApp's m_pMainWnd data member
280: BOOL CWindowApp::InitInstance()
281: {
282:   m_pMainWnd=new CAppMDIFrame();
283:   m_pMainWnd->ShowWindow(m_nCmdShow);
284:   m_pMainWnd->UpdateWindow();
285:   return TRUE;
286: }
287:
288: // application's constructor initializes and runs the app
289: CWindowApp WindowApp;
```

分　析

清单 21.2 为资源描述文件 MDIWIN1.RC 的代码。文件定义了 MDI 主窗口需要的

菜单资源,菜单有两个选项 Exit 和 MDI Children。后一个菜单项是一个弹出菜单,带几个选择项。这些选项包括:建立一个新的 MDI 子窗口、使 MDI 子窗口成为层叠式、使 MDI 子窗口成为平铺式、调整 MDI 子窗口的图标、关闭所有的 MDI 子窗口和显示 MDI 子窗口号。

清单 21.3 中列出了程序文件 MDIWIN1.CPP 的源代码,程序中声明了以下几组常量:

- 常量 MenuRezName 定义调用 Create 和 CreateClient 函数时使用的菜单名。
- 常量 Hctl 和 Wctl 用于度量 MDI 子窗口所属的静态文本和编辑控制。
- 常量 MaxWords, WordsPerLine 和 NumWords 用于在每个 MDI 子窗口中产生随机文本。

全局指针数组 Words 包含程序中使用的词汇。程序中还声明了全局函数 random 和 makeRect。随机函数产生 0 到 maxVal(函数的唯一参数)的值。makeRect 函数与本书前几天的例程中所使用的 makeRect 成员函数类似。因为 MDI 主窗口和子窗口类都需要此函数,因而把它声明为全局的,而不是在每个类中进行声明。

程序清单中声明了 3 个类:应用程序类 CWindowApp、MDI 主框架类 CAppMDIFrame 以及 MDI 子窗口类 CAppMDIChild。

应用程序类的代码看起来很象上一个程序,但有一点除外,即由 InitInstance 成员函数建立一个 MDI 框架类 CAppMDIFrame 的实例。

CAppMDIFrame 类是 CMDIFrameWnd 的子孙类,它声明了四个数据成员、一个构造函数函数、一个析构函数和一组成员函数。有两个公有和两个受保护的数据成员,公有数据成员为 bExpressClose 和 nLastMDIChild,布尔型数据成员 bExpressClose 帮助快速关闭所有窗口——本过程忽略关闭单个 MDI 子窗口时的确认要求;nLastMDIChild 成员保存 MDI 子窗口的最大索引。受保护的数据成员为 pMenu 和 nNumMDIChildren,pMenu 成员为指向附属于 MDI 主窗口的菜单的指针;nNumMDIChildren 成员存有当前的 MDI 子窗口号。

类构造函数援用 Create 成员函数,用于指定窗口的标题、风格(极大和叠加的)和 MAINMENU 菜单资源(使用全局常量 MenuRezName)。构造函数也初始化 bExpressClose, nLastMDIChild 和 nNumMDIChildren 数据成员。类的析构函数删除 pMenu 指针存取的动态菜单。

第 193 到 199 行定义了成员函数 OnCreate,它响应 Create 函数(它自己被类构造函数调用)发出的 WM_CREATE 消息,函数执行如下任务:

- 建立菜单类 CMenu 的一个新实例,并把本实例和 pMenu 数据成员联接上。
- 用 pMenu 指针调用 CMenu::LoadMenu 函数,把资源菜单 MAINMENU 附着在 MDI 主窗口上。
- 调用 CreateClient 成员函数建立不可见的 MDI 客户窗口,(CreateClient 函数的第一个参数取值,就是 OnCreate 函数的第一个参数。CreateClient 函数的第二个值是指向第一个菜单项的指针,该菜单项通过表达式 pMenu->GetSubMenu(0) 来存取)。

· 函数返回值为 0。

第 201 到 218 行定义了成员函数 CMCreateChild, 用于建立一个 MDI 子窗口。函数执行如下任务:

· 建立一个类 CAppMDIChild 的一个新的实例, 并用局部指针 pChild 存取实例。
· 调用全局函数 makeRect, 计算 MDI 子窗口的大小和位置。
· 增大数据成员 nLastMDIChild 的值。
· 使用存储在 nLastMDIChild 中的值建立 MDI 子窗口标题, 并把标题存储在局部字符串 s 中。
· 调用成员函数 CAppMDIFrame::Create, 建立 MDI 子窗口(成员函数 CAppMDIFrame::Create 的调用将提供 MDI 子窗口的标题和子窗口数目, 如果 Create 函数返回 FALSE, CMCreateChild 函数删除 CAppMDIChild 的动态实例, 减小成员 nLastMDIChild 的值, 随后退出)。
· 增加当前 MDI 子窗口数目, 子窗口数目存于数据成员 nNumMDIChildren 中。
· 调用 ShowWindow 函数, 显示新的 MDI 子窗口。

第 220 到 234 行定义了成员函数 CMCloseChildren, 用以关闭所有的 MDI 子窗口。此函数执行如下任务:

· 调用 MDIGetActive 成员函数, 获得指向当前活动 MDI 子窗口的指针(局部指针 pChild 存取活动窗口)。
· 给数据成员 bExpressClose 赋值为 TRUE。
· 用一个 while 循环语句, 从当前活动窗口开始, 关闭所有 MDI 窗口(在循环语句中, 使用 pChild 指针指向当前活动窗口, 调用 MDIDestroy 函数关闭它, 使用 MDIGetActive 函数获得新的活动子窗口的地址, 把它赋给指针 pChild)。
· 给数据成员 bExpressClose 赋值为 FALSE。

当 while 循环关闭了所有的 MDI 子窗口, MDIGetActive 函数给 pChild 指针赋值为 NULL, NULL 值终止循环。

第 237 到 243 行定义了成员函数 CMCountChildren, 用于在消息框中显示 MDI 子窗口的当前数目。此函数把存储在 nNumMDIChildren 中的值格式化成一个用于消息框显示的字符串。

第 245 到 251 行定义了成员函数 OnChildDestroy, 以响应 MDI 子窗口撤消时发出的消息。此函数减小存储在成员 nNumMDIChildren 中的值。当 nNumMDIChildren 为 0 时, 将成员 nLastMDIChild 的值重置为 0。当关闭了所有的 MDI 子窗口, 要建立的下一个 MDI 子窗口标题为"Child # 1"。

成员函数 CMCascadeChildren, CMTileChildren 和 CMArrangeIcons 分别调用继承而来的成员函数: MDICascade, MDITile 和 MDIIconArrange。

CAppMDIChild 类声明了两个公有成员, 即 TextBox 和 TextTxt, 以及保护成员 nChildNum。成员 TextBox 和 TextTxt 分别是指向建立的编辑控制和静态文本控制的指针(CAppMDIChild 实例建立其中的一个, 而不是两个)。类中还声明了一个构造函数、一

个析构函数、一个 Create 函数和一些成员函数。数据成员 nChildNum 保存用于 MDI 子窗口自我识别的子窗口号。

类构造函数把 NULL 值赋给 TextBox 和 textTxt 指针。类析构函数删除 TextBox 或 TextTxt 存取的控制的动态实例。

第 117 到 121 行定义的成员函数 Create 是继承来的 Create 函数的一个简短版本。该函数有两个参数 szText 和 nChildNumber, 函数把 nChildNumber 的值赋给成员 nChildNum, 随后调用父类的成员函数 Create。

第 123 到 157 行定义了成员函数 OnCreate, 以响应 Create 函数发出的 WM-CREATE 消息。OnCreate 函数执行如下任务:

- 为随机数发生器产生一个随机种子数。
- 使用 CString 对象 s 建立一个随机单词表(这个过程中使用了一个 for 循环, 每产生 WordsPerLine 个随机词后, 插入一个分行符)。
- 计算下一任务中要建立的编辑控制或静态文本控制的大小和位置。
- 如果 nChildNum 为偶数, 则建立一个由成员 TextBox 所存取的编辑控制; 否则, 建立一个由成员 TextTxt 存取的静态文本控制。
- 函数返回值为 0。

第 159 到 173 行定义了成员函数 OnClose, 用于关闭一个 MDI 子窗口, 可以要求也可以不要求用户确认。函数使用局部指针 pParent 存取父窗口, 即 MDI 主窗口, 这样, 就允许此函数获得 bExpressCLose 的布尔型值, 并由它决定在关闭窗口前是否提示用户。

第 175 到 179 行定义了成员函数 OnDestroy, 当一个 MDI 子窗口被删除时, 发送一条 WM- DESTROY 消息给父窗口, 即 MDI 主窗口, 这样就可以让 MDI 主窗口采取适当的行动, 比如在本例中, 是减少 nNumMDIChildren 的值。

小　　结

今天这一课介绍了多文档界面(MDI), 它是 Windows 中的一个界面标准。我们学习了如下的内容:

- MDI 应用程序的基本特性和组成。MDI 应用程序由 MDI 主窗口、不可见的 MDI 用户窗口和动态建立的 MDI 子窗口组成;
- MDI 应用程序的最基本要素是 CMDIFrameWnd 和 CMDIChildWnd 类;
- MDIFrameWnd 类管理 MDI 客户窗口、MDI 子窗口和菜单命令的执行;
- MDI 主窗口是应用程序所属的对象, 也是 MDI 子窗口的属主;
- CMDIChildWnd 类在建立 MDI 子窗口时使用。

问　与　答

问: 每一个 MDI 子窗口都应有一个 ID 值吗?

答: 对。把每个 MDI 子窗口和一个 ID 值联系起来, 可以更灵活地管理这些窗口。另

外, 在集体操作中使用 ID 值, 可以排除特定的 MDI 子窗口。

问: 可以隐藏 MDI 子窗口吗?

答: 可以。可以使用继承的成员函数 CWnd∷ShowWindow 显示和隐藏一个或多个 MDI 子窗口。

作　　业

作业部分所提供的测验题有助于巩固理解所学内容, 通过练习可以积累运用所学知识的经验。要尽量弄懂测验和练习的答案再继续下一天的课程。附录 B 的“答案”中提供了解答。

测　验

1. 对还是错?
 MDI 子窗口可以有自己的菜单。
2. 对还是错?
 MDI 子窗口可以被移到主窗口区域外;
3. 对还是错?
 MFC 库支持嵌套的 MDI 子窗口;
4. 对还是错?
 这是本书最后一道小测试题。

练　习

在 MDIWIN.CPP 中, 对每个子窗口加入命令控制, 用这些命令来切换静态文本中的大小写或编辑与每个 MDI 子窗口相关的控制。

第3周

内容复习

在即将结束 Visual C++ 编程学习的最后一周时，我们来看看猜数游戏的一个 Windows 版本。清单 R3.1 中列出了头文件 INPUTDIA.H 的源代码；清单 R3.2 为资源描述文件 INPUTDIA.RC 的源代码；清单 R3.3 列出头文件 GAME3.H 源代码；清单 R3.4 包含资源描述文件 GAM3.RC 的源代码；清单 R3.5 为程序文件 GAME3.CPP 的源代码。在目录\MSVC\VC21DAY\GAME3 中建立项目文件 GAME3.MAK，其中包括文件 GAME3.CPP，GAME3.DEF（与第 14 天的清单 14.2 中的 MINWINAD.DEF 文件相似）和 GAME3.RC。在文件 GAME3.RC 中加入 rcinclude 语句，它将自动包含 INPUTDIA.RC。

键　入　　清单 R3.1　头文件 INPUTDIA.H 的源代码

```
1: # define ID_INPUT_TXT 100
2: # define ID_INPUT_EDIT 101
```

键　入　　清单 R3.2　资源描述文件 INPUTDIA.RC

```
 1: # include < windows.h>
2: # include < afxres.h>
3: # include "inputdia.h"
4:
5: GAME DIALOG 44, 34, 200, 80
6: STYLE DS_MODALFRAME ©¦WS_CAPTION ©¦WS_VISIBLE ©¦WS_SYSMENU
7: CAPTION "Hi- Lo Guessing Game"
8: BEGIN
9:    LTEXT "Enter a string:", ID_INPUT_TXT, 20, 10, 160, 12,
10:          NOT WS_GROUP
11: EDITTEXT ID_INPUT_EDIT, 20, 25, 160, 12, ES_AUTOHSCROLL
12: DEFPUSHBUTTON "OK", IDOK, 20, 45, 65, 14, WS_GROUP
13: PUSHBUTTON "Cancel", IDCANCEL, 110, 45, 65, 14, WS_GROUP
14: END
```

键　入　　清单 R3.3　头文件 GAME3.H 的源代码

```
1: # define CM_EXIT (WM_USRE+ 100)
2: # define CM_GAME (WM_USER+ 101)
```

键　入　　清单 R3.4　资源描述文件 GAME3.RC

```
1: # include < windows.h>
2: # include < afxres.h>
3: # include "game3.h"
4:
5: MAINMENU MENU LOADONCALL MOVEABLE PURE DISCARDABLE
6: BEGIN
7:    MENUITEM "E&xit", CM_EXIT
8:    MENUITEM "&Game", CM_GAME
9: END
10:
11: rcinclude inputdia.rc
```

键　入　　清单 R3.5　程序文件 GAME3.CPP 的源代码

```
1: # include < stdlib.h>
2: # include < stdio.h>
3: # include < string.h>
4: # include < afxwin.h>
5: # include "game3.h"
6: # include "inputdia.h"
7:
8: const MaxBuffer= 81;
9:
10: // declare a global random number generating function
11: int random(int maxVal)
12: { return rand() % maxVal;}
13:
14: class CWindowApp：  public CWinApp
15: {
16: public:
17:       virtual BOOL InitInstance();
18: };
19:
20: // expand the functionality of CFrameWnd by deriving
21: // class CAppWindow
22: class CAppWindow：  public CFrameWnd
```

```
23: {
24: public:
25:
26:   CAppWindow();
27:
28: protected:
29:    // handle the Game menu item
30:   afx_msg void CMGame();
31:   afx_msg void OnExit();
32:   { SendMessage(WM_CLOSE); }
33:   // handle closing the window
34:   afx_msg void OnClose();
35:
36: // declare message map macro
37: DECLARE_MESSAGE_MAP();
38: };
39:
40: // declare a class for the line input dialog box
41: class CInputDialog : public CDialog
42: {
43: public:
44: CInputDialog(CWnd*  pParent) : CDialog("GAME", pParent) {}
45: // handle invoking the dialog box
46: int DoModal(CString& Message, CString& Text);
47: // handle initializing the dialog box
48: virtual BOOL OnInitDialog();
49: // handle the OK button
50: virtual void OnOK();
51: // access input
52: CString& GetInput() { return szText; }
53:
54: protected:
55:   CString szText;
56:   CString szMessage;
57: };
58:
59: int CInputDialog::DoModal(CString& Message, CString& Text)
60: {
61: // copy parameters to data members
62: szText= Text;
63: szMessage= Message;
64: return CDialog::DoModal();
65: }
66:
67: BOOL CInputDialog::OnInitDialog()
68: {
69: // assign the message a default input to the dialog
70: // box static and edit box controls
71: SetDlgItemText(ID_INPUT_TXT,(const char* ) szMessage);
72: SetDlgItemText(ID_INPUT_EDIT,(const char* ) szText);
73: return TRUE;
74: }
```

```
75:
76: void CInputDialog::OnOK()
77: {
78: // obtain the user's input
79: GetDlgItemText(ID_INPUT_EDIT, szText.GetBuffer(MaxBuffer),
80:             MaxBuffer);
81: szText.ReleaseBuffer();
82: EndDialog(IDOK);
83: }
84:
85: CAppWindow::CAppWindow()
86: {
87: // create the window
88: Create(NULL, "Hi- Lo Game", WS_OVERLAPPEDWINDOW,
89:       rectDefault, NULL, "MAINMENU");
90: }
91:
92: void CAppWindow::CMGame()
93: {
94: CString Message= "Enter a number between 0 and 1000";
95: CString Input= "500";
96: int nNum, nGuess;
97: int MaxIter= 11;
98: int nIter= 0;
99: BOOL bOk= TRUE;
100: CInputDialog GameDlg(this);
101:
102:
103: // reseed random- number generator
104: srand((unsigned)time(NULL));
105: nNum= random(1001);
106: // execute the opening dialog box
107: if (GameDlg.DoModal(Message, Input)==IDOK) {
108:    Input= GameDlg.GetInput();
109:    nGuess= atoi((const char* ) Input);
110:    nIter++;
111:    // loop to obtain the other guesses
112:    while (nGuess != nNum && nIter < MaxIter && bOk==TRUE) {
113:       // is the user's guess higher?
114:       Message= (nGuess > nNum)? "Enter a lower guess" :
115:                   "Enter a higher guess";
116:       bOk= (GameDlg.DoModal(Message, Input)==IDOK) ? TRUE:
117:       FALSE;
118:       Input = GameDlg.GetInput();
119:       nGuess= atoi((const char* ) Input);
120:       nIter++;
121:    }
122:
123:    // did the user guess the secret number
124:    if (nIter < MaxInter && bOk==(TRUE) {
125:      MessageBeep(0);
126:      MessageBeep(0);
```

```
127:      MessageBox("You guessed it!", "Congratulations!",
128:                 MB_OK);
129:        }
130:      else {
131:        sprintf(Message.GetBuffer(MaxBuffer),
132:                "The secret number is %d", nNum);
133:        Message.ReleaseBuffer();
134:        MessageBox((const char* ) Message, "Sorry!", MB_OK);
135:      }
136:    }
137: }
138:
139: void CAppWindow::OnClose()
140: {
141:   if (MessageBox("Want to close this application",
142:                  "Query", MB_YESNO ¦ MB_ICONQUESTION) == IDYES)
143:    DestroyWindow();
144: }
145:
146: BEGIN_MESSAGE_MAP(CAppWindow, CFrameWnd)
147:    ON_COMMAND(CM_GAME, CMGame)
148:    ON_COMMAND(CM_EXIT, OnExit)
149:    ON_WM_CLOSE()
150: END_MESSAGE_MAP()
151:
152: // Construct the CWindowApp's m_pManinWnd data member
153: BOOL CWindowApp::InitInstance()
154: {
155:   m_pMainWnd = new CAppWindow();
156:   m_pMainWnd->ShowWindow(m_nCmdShow);
157:   m_pMainWnd->UpdateWindow();
158:   return TRUE;
159: }
160:
161: // application's constructor initializes and runs the app
162: CWindowApp WindowApp;
```

图 R3.1 为程序 GAME3.EXE 的一个运行范例。

分　析

清单 R3.1 和 R3.2 一起定义了 GAME 对话框资源，该对话框提示你进行一次猜测。清单 R3.3 和 R3.4 定义了主窗口使用的菜单资源。

清单 R3.5 中的程序声明了应用程序类 CWindowApp、窗口类 CAppWindow 和输入对话框类 CInputDialog。

输入对话框类 CInputDialog 声明了一个构造函数、若干成员函数和两个 CString 数据成员。成员函数包括 DoModal, OnInitDialog, OnOK 和 GetInput。

第 59 到 65 行定义了成员函数 DoModal，它有两个 CString 引用参数：Message 和

图 R3.1　程序 GAME3.EXE 的运行范例

Text。本函数把参数 Text 和 Message 分别赋给数据成员 szText 和 szMessage。继承的成员函数 CDialog::DoModal 的返回值即为本函数的返回值。

第 67 到 74 行定义了成员函数 OnInitDialog。该函数两次调用函数 SetDlgItemText。第一次调用把成员 szMessage 中的字符串赋给输入对话框中的静态文本控制;第二次调用把成员 szText 中的字符串赋给输入对话框中的编辑控制;随后,这个布尔型函数返回 TRUE。

第 76 到 83 行定义了成员函数 OnOK,该函数把编辑控制中的文本复制到数据成员 szText 中。

成员函数 GetInput 返回数据成员 szText 中的内容。

类 CAppWindow 声明了成员函数 CMGame, OnExit 和 OnClose。函数 CMGame 开始猜数游戏并完成以下操作:

- 声明并初始化若干局部变量以及类 CString 和 CInputDilaog 的实例;
- 给随机数发生器重新置‘种子’数;
- 产生秘密数,存在变量 nNum 中;
- 执行模式对话框实例 GameDlg;当第 107 行中的 if 语句条件成立,函数执行如下的子任务:
 - 通过给实例 GameDlg 发送消息 GetInput 而获得输入字符串(结果存在字符串 Input 中)。
 - 把变量 input 中的字符串转换成一个整数,并存入变量 nGuess 中
 - 增大迭代计数器,即变量 nIter 的值。

- 执行一个 while 循环，以重复显示对话框并执行这些子任务。
- 如果用户猜中秘密数，显示一条祝贺消息；否则，在消息框中显示秘密数。

附录A

控制资源描

这一部分附录给出了菜单资源和控制资源描述的语法。虽然 Microsoft 在建立 MFC 类层次结构之前早就定义了资源描述,但正如第 19 天中所讲述的,对话框实例仍然是用资源建立的。本附录讲述以下资源:

- 菜单资源
- 对话框资源
- DIALOG 选项语句
- 通用控制资源
- 缺省控制的资源

A.1 建立菜单资源

菜单资源文件(通常以扩展名.RC 保存)中可以包含如下语句:

- MENU 语句
- MENUITEM 语句
- POPUP 语句
- MENUITEM SEPARATOR 语句
- ACCELERATORS 语句

MENU 语句定义了菜单资源的内容。MENU 语句的一般语法是:

```
menuID MENU [load options] [mem options]
BEGIN
    item definitions
END
```

menuID 是菜单资源的唯一名称或整数标识(ID)。与资源文件相关联的关键字以大写形式出现。不过,这是可选的——它们是大小写无关的。书中用了大写字母是为了易于区分资源关键字和非关键字。以下是 load 选项:

- PRELOAD——立即装入资源
- LOADONCALL——在需要时装入资源(缺省选项)

以下是 mem 选项:

- FIXED——将资源放在固定的内存位置
- MOVABLE——因为内存紧缩而需要移动资源时允许移动(缺省选项)

· DISCARDABLE——当不再需要该资源时丢弃它(亦为缺省选项)

MENUITEM 语句定义了实际菜单项的名字和属性。MENUITEM 语句的一般语法是:

```
MENUITEM text, result [, option list]
```

text 域应为一个字符串(括在双引号内),它指明菜单项的名字。若要为菜单项定义一个热键,可以将 & 字符放在想使之成为热键的字母的前面(菜单选择是大小写无关的)。若要在菜单名中显示 & 字符,则可以使用连续两个 & 字符(即 &&)。还可以在菜单项名字中包含\t 序列以插入水平制表符。若插入\a 序列则可以将其后的所有文本右对齐。

result 是个整数,通常它是一个 CM- XXXX 常量,代表由菜单项发出的命令。

option list 中可以包含以下各项:

· CHECKED——在菜单项旁边显示一个选中标记
· GRAYED——以灰色显示菜单项以表明菜单项是活动的
· HELP——在菜单项的左边放置一垂直分隔条
· INACTIVE——显示菜单项,但它不能被选中。此选项通常与 GRAYED 选项结合起来使用以在视觉上更好地表示该菜单项是不活动的
· MENUBARBREAK——将菜单项放到新行上作为静态菜单栏项。对于弹出式菜单项(下面要讲到),将新列和旧列用垂直线隔开
· MENUBREAK——将菜单项放到新行上作为静态菜单栏项。对于弹出式菜单项,将菜单项放到新的一列上,在新旧列之间没有分隔线

POPUP 语句表是开始定义弹出式菜单。POPUP 语句的一般语法是:

```
 POPUP text [, option list]
BEGIN
    item definitions
END
```

text 和 option list 域与 MENUITEM 语句中的相应域类似。item definitons(选项定义)由 MENUITEM 或其他 POPUP 语句组成。用后者可以建立嵌套菜单。

MENUITEM SEPARATOR 语句是 MENUITEM 语句的特殊形式,它建立一个非活动的菜单项并在两个活动菜单项之间显示一条分隔线。

ACCELERATORS 语句定义 MFC 应用程序的一个或多个加速键。

新　词

加速键(accelerator)
加速键是为了使应用程序用户快速地选择菜单项并执行特定任务而定义的按键。

ACCELERATORS 语句的一般语法是:

```
accTableName ACCELARATORS
BEGIN
    event, idValue, [type] [NOINVERT] [ALT] [SHIFT] [CONTROL]
END
```

accTableName 域为加速键资源定义了唯一名称和整数标识(ID)以将它和其他类型资源相区别。event 域指定加速键的按键,它可以取以下值:

- 括在双引号内的一个 ASCII 字符。可以将^ 符号放在字符前面以表示控制字符,这种情况下就不再需要 type 域了。
- 一个指定字符的 ASCII 码的整数值。这种情况下,type 域必须是关键字 ASCII。
- 一个代表虚拟键的整数值。这种情况下,type 域必须是关键字 VIRKEY。

idValue 域是用来标识加速键的整数值。type 域仅当 event 域是一个 ASCII 字符码或虚拟键时才需要。

NOINVERT 选项禁止在使用加速键时突出显示菜单。ALT, SHIFT 和 CONTROL 选项分别表示当 Alt, Shift 和 Control 键被按下时激活加速键。以下是一个加速键资源的例子:

```
"EditKeys" ACCELARATORS
BEGIN
    "h",        IDDHEADING                      ; the H key
    "H",        IDDHOLD                         ; the Shift- H keys
    "^B",       IDDBOLD                         ; the Control- B keys
    64,         IDDADD                          ; the Shift^ keys
    97,         IDDAPPEND                       ; the A key
    "s",        IDDSEARCH, ALT                  ; the Alt- S keys
    VK_F7,      IDDSAVE, VIRKEY                 ; The F7 function key
    VK_F2,      IDDLOAD, SHIFT, VIRKEY          ; The Shift- 2 keys
    VK_F3,      IDDSAVEAS, CONTROL, VIRKEY      ; The Control- F3 keys
    VK_F1,      IDDNEW, ALT, SHIFT, VIRKEY      ; The Alt- Shift- F1
                                                ; keys
END
```

A.2 对话框资源

DIALOG 语句定义了在 Windows 程序中用来建立对话框的资源。DIALOG 语句的一般语法是:

```
nameID DIALOG [load-option][mem-option] x, y, width, height
[option-statements]
BEGIN
    control-statements
END
```

nameID 是对话框资源的唯一名称或整数标识(ID)。与资源文件相关联的关键字以大写形式出现。不过,这是可选的——它们是大小写无关的。书中用了大写字母是为了易于区分资源关键字和非关键字。以下是 load 选项:

- PRELOAD——立即装入资源
- LOADONCALL——在需要时装入资源(缺省选项)

以下是 mem 选项:

- FIXED——将资源放在固定的内存位置
- MOVABLE——因为内存紧缩而需要移动资源时允许移动(缺省选项)
- DISCARDABLE——当不再需要该资源时丢弃它(亦为缺省选项)

x 和 y 参数指定对话框左上角的位置。width 和 height 参数定义了对话框的尺寸。下面的例子中定义了对话框资源,这是从文件 DIALOG1.RC 中摘录下来的:

```
# include < windows.h>
# include < afxres.h>

NEW DIALOG DISCARDABLE LOADONCALL PURE MOVEABLE 30, 50, 200, 100
STYLE WS- POPUP ©¦DS- MODALFRAME
CAPTION "Message"
BEGIN
    CTEXT "Exit the application?", 1, 10, 10, 170, 15
   CONTROL "OK", IDOK, "BUTTON", WS- CHILD ©¦WS- VISIBLE ©¦
     WS- TABSTOP ©¦BS- DEFPUSHBUTTON, 20, 50, 70, 15
CONTROL "Cancel", IDCANCEL, "BUTTON", WS- CHILD ©¦WS- VISIBLE ©¦
        WS- TABSTOP ©¦BS- PUSHBUTTON, 110, 50, 70, 15
END
```

A.3　DIALOG 选项语句

DIALOG 选项语句指定对话框的特殊属性,如风格、标题和菜单。这些语句是可选的。如果在对话框资源定义中不带任何选项语句,则对话框具有缺省的属性。DIALOG 选项语句包括以下各项:

- STYLE
- CAPTION
- MENU
- CLASS
- FONT

以下各小节讲述这些对话框属性。

STYLE 语句

STYLE 语句指定对话框的窗口风格。这一属性表明对话框是否子窗口或弹出式窗口。对话框的缺省风格有 WS- POPUP, WS- BORDER 和 WS- SYSMENU 风格。STYLE 语句的一般语法是:

```
STYLE style
```

style 参数是一个整数值,它由风格属性进行按位或(OR)形成。STYLE 选项语句举例:

```
STYLE WS- POPUP ©¦DS- MODALFRAME
```

CAPTION 语句

CAPTION 语句定义了对话框标题。如果对话框有标题栏,则在其中显示标题。缺省情况下,标题是个空串。CAPTION 语句的一般语法是:

```
CAPTION title
```

title 参数是个字符串。CAPTION 语句举例如下:

```
CAPTION "Replace Text"
```

MENU 语句

MENU 语句指定附着在对话框上的菜单。缺省情况下,对话框没有菜单。MENU 语句的一般语法是:

```
MENU menuName
```

menuName 参数是菜单资源的名字或代码。MENU 语句举例如下:

```
# include < windows.h>
# include < afxres.h>

YesNo MENU LOADONCALL MOVEABLE PURE DISCARDABLE
BEGIN
    MENUITEM "&Ok", IDOK
    MENUITEM "&Cancel", IDCANCEL
END

NEW DIALOG DISCARDABLE LOADONCALL PURE MOVEABLE 30, 50, 200, 100
STYLE WS_POPUP ©¦DS_MODALFRAME
CAPTION "Message"
MENU YesNo
BEGIN
   CTEXT "Exit the application?", 1, 10, 10, 170, 15
   CONTROL "OK", IDOK, "BUTTON", WS_CHILD ©¦WS_VISIBLE ©¦
     WS_TABSTOP ©¦BS_DEFPUSHBUTTON, 20, 50, 70, 15
   CONTROL "Cancel", IDCANCEL, "BUTTON", WS_CHILD ©¦WS_VISIBLE ©¦
     WS_TABSTOP ©¦BS_PUSHBUTTON, 110, 50, 70, 15
END
```

CLASS 语句

CLASS 语句指定对话框的 Windows 注册类(而不是 MFC 库类)。CLASS 语句的一般语法是:

```
CLASS className
```

className 参数定义注册类名的整数或串。CLASS 语句举例如下:

```
CLASS "ChitChat"
```

FONT 语句

FONT 语句指定 Windows 用来在对话框中绘制文本的字体。指定的字体必须已经从 WIN.INI 或通过调用 API 函数 LoadFont 装入。FONT 语句的一般语法是:

FONT pointSize，typeface

pointSize 参数是一个整数，它以点为单位指定字体的大小。typeface 参数是一个字符串，表示字体的名称。FONT 语句举例如下：

FONT 10, "Helv"

A.4　对话框控制资源

资源描述支持两种控制资源。第一种是 CONTROL 语句，它给出了声明控制资源的一般方法；另一种是可修饰的缺省控制资源，这些资源使用的语句中带有描述其定义的控制的关键字。例如，RADIOBUTTON 语句定义了圆按钮的资源。下一节讲述 CONTROL 语句，再下一节讲述定义特定控制的资源的语句。

A.5　一般控制资源

用 CONTROL 语句可以定义对话框所属的任何标准的或用户定义控制的资源。CONTROL 语句的一般语法如下：

CONTROL text, id, class, style, x, y, width, height

text 参数指定控制中所出现的文本的字符串。

id 参数声明了控制的唯一标识(ID)。

> **注　意**　class 参数是一个表示控制的 Windows 注册类名的字符串。表 A.1 中列出了各种标准控制的 Windows 注册类名。注意下压按钮、选中框、圆按钮和分组框都具有同一注册类名！那么它们彼此如何区分？这就要靠 style 参数。

> **注　意**　style 参数通常是一个按位或的表达式，它与控制相关联的所有风格。没有缺省风格！

x，y，width 和 height 参数指定控制的位置和尺寸。这些参数一般都是整型常量。还可以用加法运算符建立简单的表达式。

表 A.1　各种标准控制的 Windows 注册类名

控制	注册类名
选中框	BUTTON
组合框	COMBOBOX
编辑框	EDIT
分组框	BUTTON
列表框	LISTBOX
下压按钮	BUTTON
圆按钮	BUTTON
滚动条	SCROLLBAR
静态文本	STATIC

以下是使用 CONTROL 语句建立对话框控制的一些例子,这些例子是从资源文件 DIALOG6.RC 中摘录下来的:

```
ID_DIALOG DIALOG DISCARDABLE LOADONCALL PURE MOVEABLE 10, 10, 200, 150
STYLE WS_POPUP ©¦WS_CLIPSIBLINGS ©¦WS_CAPTION ©¦WS_SYSMENU ©¦
DS_MODALFRAME
CAPTION "Controls Demo"
BEGIN
    CONTROL "Find" ID_FIND_TXT, "STATIC", WS_CHILD ©¦WS_VISIBLE ©¦
      SS_LEFT, 20, 10, 100, 15

    CONTROL "", ID_FIND_CMB, "COMBOBOX", WS_CHILD ©¦WS_VISIBLE ©¦
      WS_BORDER ©¦WS_TABSTOP ©¦CBS_DROPDOWN, 20, 25, 100, 50

    CONTROL "Replace", ID_REPLACE_TXT, "STATIC", WS_CHILD ©¦WS_VISIBLE ©¦
       SS_LEFT, 20, 45, 100, 15

    CONTROL "", REPLACE_CMB, "COMBOBOX", WS_CHILD ©¦WS_VISIBLE ©¦
      WS_BORDER ©¦WS_TABSTOP ©¦CBS_DROPDOWN, 20, 60, 100, 50

   CONTRO "Scope", ID_SCOPE_GRP, "BUTTON", WS_CHILD ©¦WS_VISIBLE
      ©¦WS_GROUP ©¦BS_GROUPBOX, 20, 80, 90, 15

   CONTROL "Global", ID_GLOBAL_RBT, "BUTTON", WS_CHILD ©¦WS_VISIBLE
     ©¦WS_TABSTOP ©¦BS_AUTORADIOBUTTON, 30, 90, 50, 15

    CONTROL "Selected Text", ID_SELTEXT_RBT, "BUTTON", WS_CHILD ©¦
      WS_VISIBLE ©¦WS_TABSTOP ©¦BS_AUTOCHECKBOX, 30, 105, 60, 15

    CONTROL, "Case Sensitive", ID_CASE_CHK, "BUTTON", WS_CHILD ©¦
     WS_VISIBLE ©¦WS_TABSTOP ©¦BS_AUTOCHECKBOX, 20, 130, 80, 15

   CONTROL "Whole Word", ID_WHOLEWORD_CHK, "BUTTON", WS_CHILD ©¦
     WS_VISIBLE ©¦WS_TABSTOP ©¦BS_AUTOCHECKBOX, 100, 130, 80, 15

   CONTROL "&OK", IDOK, "BUTTON", WS_CHILD ©¦WS_VISIBLE ©¦WS_TABSTOP
     ©¦BS_DEFPUSHBUTTON, 120, 90, 30, 20
```

```
    CONTROL "&Cancel", IDCANCEL, "BUTTON", WS_CHILD ©¦WS_VISIBLE ©¦
      WS_TABSTOP, 160, 90, 30, 20
END
```

A.6　LTEXT 语句

LTEXT 语句定义了文本左对齐的静态文本控制的资源。LTEXT 语句的一般语法是:

```
LTEXT text, id, x, y, width, height [, style]
```

text 参数指定静态控制文本。该文本可以包括一个 & 字符以便给热键字符加下划线。id 参数定义静态控制的标识(ID)。x, y, width 和 height 参数指定控制的位置和尺寸。可选的 style 参数指定资源的附加风格。缺省的风格是 SS_LEFT 和 WS_GROUP。style 参数可以是 WS_TABSTOP, WS_GROUP 或二者兼而有之。

text 参数中的字符以左对齐方式显示。如果整个文本在指定宽度内显示不下,超出的字符将折返到下一行开始处显示。

LTEXT 语句举例:

```
LTEXT "Current Drive:", ID_DRIVE_TXT, 10, 10, 50, 10
LTEXT "Current Dir:", ID_DIR_TXT, 10, 50, 50, 10, WS_TABSTOP ©¦WS_GROUP
```

A.7　RTEXT 语句

RTEXT 语句定义了文本左对齐的静态文本控制的资源。RTEXT 语句的一般语法是:

```
RTEXT text, id, x, y, width, height [, style]
```

text 参数指定静态控制文本。该文本可以包括一个 & 字符以便给热键字符加下划线。id 参数定义静态控制的标识(ID)。x, y, width 和 height 参数指定控制的位置和尺寸。可选的 style 参数指定资源的附加风格。缺省的风格是 SS_RIGHT 和 WS_GROUP。style 参数可以是 WS_TABSTOP, WS_GROUP 或二者兼而有之。

text 参数中的字符以右对齐方式显示。如果整个文本在指定宽度内显示不下,超出的字符将折返到下一行,且同样以右对齐方式显示。

RTEXT 语句举例:

```
RTEXT "Current Drive:", ID_DRIVE_TXT, 70, 10, 50, 10
```

A.8　CTEXT 语句

CTEXT 语句定义了文本居中的静态文本控制的资源。CTEXT 语句的一般语法是:

```
CTEXT text, id, x, y, width, height [, style]
```

text 参数指定静态控制文本。该文本可以包括一个 & 字符以便给热键字符加下划线。id 参数定义静态控制的标识(ID)。x，y，width 和 height 参数指定控制的位置和尺寸。可选的 style 参数指定资源的附加风格。缺省的风格是 SS_CENTER 和 WS_GROUP。style 参数可以是 WS_TABSTOP，WS_GROUP 或二者兼而有之。

text 参数中的字符以右对齐方式显示。如果整个文本在指定宽度内显示不下，超出的字符将折返到下一行，且同样以居中方式显示。

CTEXT 语句举例：

```
CTEXT "Current Drive:", ID_DRIVE_TXT, 10, 10, 50, 10
```

A.9　CHECKBOX 语句

CHECKBOX 语句定义了注册类为 BUTTON 的选中框控制的资源。CHECKBOX 语句的一般语法是：

```
CHECKBOX text, id, x, y, width, height [, style]
```

text 参数指定控制的标题。该文本可以包括一个 & 字符以便给热键字符加下划线。id 参数定义静态控制的标识(ID)。x，y，width 和 height 参数指定控制的位置和尺寸。可选的 style 参数指定资源的附加风格。缺省的风格是 BS_CHECKBOX 和 WS_TABSTOP。style 参数可以是 WS_DISABLED，WS_TABSTOP 或二者兼而有之。

CHECKBOX 语句举例：

```
CHECKBOX "Case-Sensitive:", ID_CASE_CHECK, 10, 10, 100, 10
```

A.10　PUSHBUTTON 语句

PUSHBUTTON 语句定义了注册类为 BUTTON 的下压按钮控制的资源。PUSHBUTTON 语句的一般语法是：

```
PUSHBUTTON text, id, x, y, width, height [, style]
```

text 参数指定控制的标题。该文本可以包括一个 & 字符以便给热键字符加下划线。id 参数定义静态控制的标识(ID)。x，y，width 和 height 参数指定控制的位置和尺寸。可选的 style 参数指定资源的附加风格。缺省的风格是 BS_PUSHBUTTON 和 WS_TABSTOP。style 参数可以是 WS_DISABLED，WS_TABSTOP，WS_GROUP 或这三种风格之间以按位或(OR)方式进行任意组合后的值。

PUSHBUTTON 语句举例如下：

```
PUSHBUTTON "Calculate", ID_CALC_BTN, 10, 10, 100, 10, WS_DISABLED
```

A.11　DEFPUSHBUTTON 语句

DEFPUSHBUTTON 语句定义了注册类为 BUTTON 的缺省下压按钮控制的资源。DEFPUSHBUTTON 语句的一般语法是:

```
DEFPUSHBUTTON text, id, x, y, width, height [, style]
```

text 参数指定控制的标题。该文本可以包括一个 & 字符以便给热键字符加下划线。id 参数定义静态控制的标识(ID)。x, y, width 和 height 参数指定控制的位置和尺寸。可选的 style 参数指定资源的附加风格。缺省的风格是 BS_DEFPUSHBUTTON 和 WS_TABSTOP。style 参数可以是 WS_DISABLED, WS_TABSTOP, WS_GROUP 或这三种风格之间以按位或(OR)方式进行任意组合后的值。

DEFPUSHBUTTON 语句举例如下:

```
DEFPUSHBUTTON "Calculate", ID_CALC_BTN, 10, 10, 100, 10
```

A.12　LISTBOX 语句

LISTBOX 语句定义了注册类为 LISTBOX 的列表框控制的资源。LISTBOX 语句的一般语法是:

```
LISTBOX id, x, y, width, height [, style]
```

id 参数定义列表框控制的标识(ID)。x, y, width 和 height 参数指定控制的位置和尺寸。可选的 style 参数指定资源的附加风格。缺省的风格是 LBS_NOTIFY, WS_VSCROLL 和 WS_BORDER。style 参数可以是 WS_BORDER, WS_VSCROLL 或二者兼而有之。

LISTBOX 语句举例:

```
LISTBOX ID_OPERAND_LST, 10, 10, 100, 100
```

A.13　GROUPBOX 语句

GROUPBOX 语句定义了注册类为 BUTTON 的分组框控制的资源。GROUPBOX 语句的一般语法是:

```
GROUPBOX text, id, x, y, width, height [, style]
```

text 参数指定控制的标题。该文本可以包括一个 & 字符以便给热键字符加下划线。id 参数定义静态控制的标识(ID)。x, y, width 和 height 参数指定控制的位置和尺寸。可选的 style 参数指定资源的附加风格。缺省的风格是 BS_GROUPBOX 和 WS_TABSTOP。style 参数可以是 WS_TABSTOP, WS_DISABLED 风格或二者兼而有之。

GROUPBOX 语句举例:

```
GROUPBOX "Angle:", ID_ANGLE_GRP, 10, 10, 200, 200
```

A.14　RADIOBUTTON 语句

RADIOBUTTON 语句定义了注册类为 BUTTON 的圆按钮控制的资源。RADIOBUTTON 语句的一般语法是:

```
RADIOBUTTON text, id, x, y, width, height [, style]
```

text 参数指定控制的标题。该文本可以包括一个 & 字符以便给热键字符加下划线。id 参数定义静态控制的标识(ID)。x, y, width 和 height 参数指定控制的位置和尺寸。可选的 style 参数指定资源的附加风格。缺省的风格是 BS_RADIOBUTTON 和 WS_TABSTOP。style 参数可以是 WS_TABSTOP, WS_DISABLED, WS_GROUP 风格或这三种风格之间以按位或(OR)方式进行任意组合后的值。

RADIOBUTTON 语句举例:

```
RADIOBUTTON "Degrees:", ID_DEGREES_RBT, 10, 10, 100, 10
```

A.15　EDITTEXT 语句

EDITTEXT 语句定义了注册类为 EDIT 的编辑框控制的资源。EDITTEXT 语句的一般语法是:

```
EDITTEXT id, x, y, width, height [, style]
```

id 参数定义编辑框控制的标识(ID)。x, y, width 和 height 参数指定控制的位置和尺寸。可选的 style 参数指定资源的附加风格。缺省的风格是 WS_TABSTOP, ES_EDIT 和 WS_BORDER。style 参数可以是 WS_TABSTOP, WS_GROUP, WS_VSCROLL, WS_HSCROLL, WS_DISABLED 或这几种风格之间以按位或(OR)方式进行任意组合后的值。

EDITTEXT 语句举例:

```
EDITTEXT ID_INPUT_BOX, 10, 10, 200, 200
```

A.16　COMBOBOX 语句

COMBOBOX 语句定义了注册类为 COMBOBOX 的组合框控制的资源。COMBOBOX 语句的一般语法是:

```
COMBOBOX id, x, y, width, height [, style]
```

id 参数定义组合框控制的标识(ID)。x, y, width 和 height 参数指定控制的位置和尺寸。可选的 style 参数指定资源的附加风格。缺省的风格是 WS_TABSTOP 和 CBS_SIMPLE。style 参数可以是 WS_TABSTOP, WS_GROUP, WS_VSCROLL, WS_

DISABLED 或这几种风格之间以按位或(OR)方式进行任意组合后的值。

COMBOBOX 语句举例:

```
COMBOBOX ID_INPUT_BOX, 10, 10, 200, 200
```

A.17　SCROLLBAR 语句

SCROLLBAR 语句定义了注册类为 SCROLLBAR 的滚动条控制的资源。SCROLLBAR 语句的一般语法是:

```
SCROLLBAR id, x, y, width, height [, style]
```

id 参数定义滚动条控制的标识(ID)。x，y，width 和 height 参数指定控制的位置和尺寸。可选的 style 参数指定资源的附加风格。缺省的风格是 SBS_HORZ。style 参数可以是 WS_TABSTOP，WS_GROUP，WS_DISABLED 或这几种风格之间以按位或(OR)方式进行任意组合后的值。

SCROLLBAR 语句举例:

```
SCROLLBAR ID_INDEX_SCR, 10, 10, 20, 200
```

小　　结

本附录简要地介绍了用来建立对话框及其控制的资源语句,主要内容如下:

- DIALOG 语句定义对话框资源;
- DIALOG 选项语句包括 STYLE，CAPTION，MENU，CLASS 和 FONT 语句;
- CONTROL 语句定义标准的以及用户定义的控制的资源;
- LTEXT 语句定义左对齐静态文本控制的资源;
- RTEXT 语句定义右对齐静态文本控制的资源;
- CTEXT 语句定义居中静态文本控制的资源;
- CHECKBOX 语句定义选中框控制的资源;
- PUSHBUTTON 语句定义下压按钮控制的资源;
- DEFPUSHBUTTON 语句定义缺省下压按钮控制的资源;
- LISTBOX 语句定义列表框控制的资源;
- GROUPBOX 语句定义分组框控制的资源;
- RADIOBUTTON 语句定义圆按钮控制的资源;
- EDITTEXT 语句定义编辑框控制的资源;
- COMBOBOX 语句定义组合框控制的资源;
- SCROLLBAR 语句定义滚动条控制的资源。

附录B

答　案

注　意　由于篇幅限制，不是所有练习都给出了答案。

第 1 天

测　验

1. 程序产生的字符串为"C+ +　in 21 Days?"。
2. 因为 cout 语句在注释内，程序不产生输出。函数 main 只是简单地返回 0。
3. cout 语句遗失分号。

练　习

```
    // Exercise program
# include < iostream.h>
main ()
{
  cout < <  "I am C+ +  Programmer";
  return 0;
}
```

第 2 天

测　验

1. 下表中说明了哪个标识符是合法的、哪个是非法的以及为什么。

标识符	是否合法	原因(如非法)
numFiles	是	
n0Distance_02_Line	是	
0Weight	否	以数字开始
Bin Number	否	内含空格
static	否	保留关键字
Static	是	

2. 程序输出为：

```
a = 10 and b = 3
```

函数 swap 未能交换 a 和 b 的值,因为它只交换了其值的一个拷贝。

3. 程序输出为:

```
a = 3 and b = 3
```

因为函数 swap 使用了引用参数,所以成功地交换了 a 和 b 的值。接着,参数 i 和 j 的值改变,超出了函数本身的作用域。

4. 函数 inc 第 2 个版本中有一个缺省变元,因而在使用时编译器无法判断调用哪个版本的 inc。对于这样的函数,编译器给出一个编译时错误标记。
5. 因为第 2 个参数有一个缺省值,所以第 3 个也必须有缺省值。以下是函数 volume 的一种正确定义形式:

```
double volume(double length, double width = 1, double height = 1)
{
  return length * width * height
}
```

6. 参数 i 是小写字母,但函数在赋值语句中使用了大写字母 I。编译器提示标识符 I 未定义。
7. 函数 main 需要函数 sqr 的原型。正确的程序为:

```
# inclue < iostream.h>

// declare prototype of function sqr
double sqr(double);

main()
{
  double x = 5.2;

  cout < < x < < "^ 2 = " < < sqr(x);
  return 0;
}

double sqr(double x)
{ return x * x; }
```

练　习

下面的程序 OVERLOD1.CPP 供参考:

```
// C+ + program illustrates function overloading
// and default arguments

# include < iostream.h>

// inc version for int types
void inc(int& i, int diff = 1)
{
  i = i + diff;
```

```
}

// inc version for double types
void inc (double& x, double diff = 1)
{
  x = x + diff;
}

// inc version for char types
void inc (char& c, int diff = 1)
{
  c = c + diff;
}

main()
{
  char c = 'A';
  int i = 10;
  double x = 10.2;

  // display initial values
  cout << "c = " << c << "\n"
       << "i = " << i << "\n"
       << "x = " << x << "\n";
  // invoke the inc functions using default arguments
  inc(c);
  inc(i);
  inc(x);
  // display updated values
  cout << "After using the overloaded inc function\n";
  cout << "c = " << c << "\n"
       << "i = " << i << "\n"
       << "x = " << x << "\n"
  return 0;
}
```

第 3 天

测　验

1. 输出为:

```
12
   8
   2
   3.64851
   150.5
```

2. 输出为:

```
12
   8
```

```
2
```

3. **输出为**:

```
12
27
```

4. **输出为**:

```
TRUE
TRUE
TRUE
FALSE
```

练　习

1. **下面** max **函数供参考**:

```
int max(int i, int j)
{
  return (i > j) ? i : j;
}
```

2. **下面的** min **函数供参考**:

```
int min(int i, int j)
{
  return (i > 0) ? i : j;
}
```

3. **下面的** abs **函数供参考**:

```
int abs(int i)
{
  return (i > 0) ? i: - i;
}
```

4. **下面的** isodd **函数供参考**:

```
int isodd(int i)
{
  return (i % 2 ! = 0) ? 1 : 0;
}
```

第 4 天

测　验

1. **输出语句中不能含有插入运算符“** > > **”,正确的语句如下**:

```
cout < < "Enter a number";
cin > > x;
```

2. **因为第一项和最后一项都出现变量** X, **所以最后一个数是将覆盖掉第一个数。**

练　习

1. 下面的 OUT3.CPP 程序供参考:

```
// C++ program uses the printf function for formatted output
# inlude <stdio.h>
# include <math.h>

main()
{
   double x;

   // display table heading
   printf("        X          Sqr(X)\n");
   printf("-----------------------\n);
   x = 2;
   printf("     %3.0lf      %3.4lf\n",x,sqrt(x));
   x++;
   printf("     %3.0lf      %3.4lf\n",x,sqrt(x));
   x++;
   printf("     %3.0lf      %3.4lf\n",x,sqrt(x));
   x++;
   printf("     %3.0lf      %3.4lf\n",x,sqrt(x));
   x++;
   printf("     %3.0lf      %3.4lf\n",x,sqrt(x));
   x++;
   printf("     %3.0lf      %3.4lf\n",x,sqrt(x));
   x++;
   printf("     %3.0lf      %3.4lf\n",x,sqrt(x));
   x++;
   printf("     %3.0lf      %3.4lf\n",x,sqrt(x));
   x++;
   printf("     %3.0lf      %3.4lf\n",x,sqrt(x));
   x++;
   printf("     %3.0lf      %3.4lf\n",x,sqrt(x));
   return 0;
}
```

2. 下面的 OUT4.CPP 程序供参考:

```
// C++ program which displays octal and hexadecimal integers

# include <iostream.h>
# include <stdio.h>

main()
{
  long i;
  cout << "Enter an integer: ";
  cin >> i;
  printf("%ld = %lX (hex) = %lo (octal)\n", i, i, i);
  return 0;
}
```

第 5 天

测　验

1. 简化版本为:

```
if (i > 0 && i < 10)
  cout << "i = " << i << "\n";
```

2. 简化版本为:

```
if (i > 0) {
  j = i * i;
  cout << "j = " << j << "\n";
}
else if (i < 0) {
  j = 4 * i;
  cout << "j = "<< j << "\n";
}
else {
  j = 10 + i;
  cout << "j = " << j << "\n";
}
```

3. 不正确。

当变量 i 值在- 10 与- 1 之间时,两个 if 语句的子句都被执行,这时将执行所有赋值语句。相反,在假设等价的 if-else 语句中,if 和 else 字句中的语句不可能都执行。

4. 可简化为:

```
if (i > 0 && i < 100)
  j = i * i;
else if (i > = 100)
  j = i;
else
  j = 1;
```

注意这里去掉了原来的第一个 else if 子句,因其测定条件为第一个测试条件的子集,所以第一个 else if 的条件永远不可能被检测到,因而有关的赋值语句也从未执行过。这种代码被称为死码(dead code)。

5. 测试条件始终为假,因而子句中的语句不可能被执行。这是又一例死码。

练　习

1. 以下的 IF5.CPP 程序供参考:

```
// C+ + program to solve quadratic equtaion

# include < iostream.h>
# include < math.h>
```

```
main()
{
  double A, B, C discrim, root1, root2, twoA;

  cout << "Enter coefficients for equation A* X^ 2+ B* X+ C\n";
  cout << "Enter A:";
  cin >> A;
  cout >> "Enter B:";
  cin >> B;
  cout << "Enter C:";
  cin >> C;
  if (A != 0) {
    twoA = 2 * A;
    discrim = B* B - 4* A* C;
    if (discrim > 0) {
      root1= (- B+ sqrt(discrim))/twoA;
      root2= (- B- sqrt(discrim))/twoA;
      cout << "root1= " << root1 << "\n";
      cout << "root2= " << root2 << "\n";
    }
    else if (discrim < 0) {

      discrim= - discrim;
      cout << "root1= ("<< - B/twoA
           << ") + i (" << sqrt(discrim)/twoA << ")\n";
      cout << "root2= ("<< - B/twoA
           << ") - i (" << sqrt(discrim)/twoA << ")\n";
    }
    else {
      root1= - B/2/A;
      root2= root1;
      cout << "root1= "<< root1<< "\n";
      cout << "root2= 2"<< root2<< "\n";
    }
}
else
   cout << "root= "<< (- C/B)<< "\n";

return 0;

}
```

2. 以下的 SWITCH2.CPP 程序供参考:

```
// C++ program which uses the switch statemet to implement
// a simple four-function calculator program

# include < iostream.h>

const in TRUE= 1;
const in FALSE= 0;

main()
```

```
{
  double x, y, z;
  char op;
  int error= FALSE;

  cout < < "Enter the first operand: ";
  cin > > x;
  cout < < "Enter the operator: ";
  cin > > op;
  cout < < "Enter the second operand: ";
  cin > > y;
  switch(op) {
    case '+' :
      z= x+ y;
      break;
    case '-' :
      z= x- y;
      break;
    case '*' :
      z= x* y;
      break;
    case '/' :
      if (y ! = 0)
        z= x/ y;
      else
        error= TRUE;
      break;
    default:
      error= TRUE;
  }
  if (! error)
    cout < < x < < " " < < op < < " " < < y < < " = " < < z < < "\n";
  else
     cout < < "Bad operator or division-by-zero error\n";
  return 0;
}
```

第 6 天

测　验

1. 循环中的语句未能改变 I 的值,因而测试条件始永远为真,循环无休止地迭代。
2. 程序输出结果为 3，5，7。
3. 程序输出为无休止地显示值 3,无限循环原因是循环控制变量未增加。
4. 嵌套 for 循环使用相同的循环控制变量,程序不能运行。
5. 两个 for 循环都声明变量 i 作为它们的控制变量,这种重复将导致编译器产生一个出错信息。
6. while 循环的条件永远为真,循环将无休止迭代。
7. 程序缺少一条将变量 factorial 显式初始化为 1 的语句。没有这条语句,程序自动

初始化变量 factorial 为 0——而这是一个错误的值,因而 for 循环在每次迭代结束时都给变量 factorial 赋值为 0。正确的代码是:

```
int n
double factorial = 1;
cout << "Enter positive integer : ";
cin >> n;
for (int i = 1; i <= n; i++)
factorial *= i;
cout << n << "! = " << factorial;
```

练　习

1. 下面的 FOR5.CPP 程序供参考:

```
// Program calculates a sum of odd integers in
// the range of 11 to 121

# include <iostream.h>

const int FIRST = 11;
const int LAST = 121;

main()
{
  double sum = 0;
  for (int i = FIRST; i <= LAST; i += 2)
    sum += (double)i;
  cout << "Sum of odd integers from"
       << FIRST << "to" << LAST << "= "
       << sum << "\n";
  return 0;
}
```

2. 下面的 WHILE2.CPP 程序供参考:

```
// Program calculates a sum of squared odd integers in
// the range of 11 to 121

# include <iostream.h>

const int FIRST = 11;
const int LAST = 121;

main()
{
  double sum = 0;
  int i = FIRST;
  while (i <= LAST) {
    sum += double(i * i++);
  }

  cout << "Sum of odd integers from"
```

```
        << FIRST << "to" << LAST << "= "
        << sum << "\n";
  return 0;
}
```

3. 以下的 DOWHILE2.CPP 程序供参考:

```
// Program calculates a sum of squared odd integers in
// the range of 11 to 121

# include < iostream.h>

const int FIRST = 11;
const int LAST = 121;
main()
{
  double sum = 0;
  int i = FIRST;
  do {
    sum += double(i * i++);
  } while (i <= LAST);
  cout << "Sum of odd integers from"
        << FIRST << "to" << LAST << "= "
        << sum << "\n";
  return 0;
}
```

第 7 天

测　验

1. 程序显示 0 到 4 的阶乘数:

```
X[0] = 1
X[1] = 1
X[2] = 2
X[3] = 6
X[4] = 24
```

2. 程序显示 0 到 4 的平方根:

```
X[0] = 0
X[1] = 1
X[2] = 1.41421
X[3] = 1.73205
X[4] = 2
```

3. 第一个 for 循环应在 1 到 MAX-1 之间而不是 0 到 MAX-1 之间进行迭代。第一次循环迭代的索引越界。

练　习

以下的 ARRAY7.CPP 程序供参考:

```
// C++ program that sorts arrays using the Comb sort method

# include < iostream.h>

const int MAX= 10;
const int TRUE= 1;
const int FALSE= 0;

int obtainNumData()
{
  int m;
  do { //obtain number of data points
    cout < < "Enter number of data points [2 to "
         < < MAX < < "]:";
    cin > > m;
    cout < < "\n";
  } while (m< 2    m > MAX);
  return m;
}

void inputArray(int intArr[], int n)
{
  // prompt user for data
  for (int i= 0;i< n;i++ ) {
    cout < < "arr[" < < i < < "]:";
    cin > > intArr[i];
  }
}

void showArray(int intArr[],int n)
{
  for (int i= 0 ; i < n;i++ ) {
    cout.width (5);
    cout < < intArr[i] < < " ";
  }
  cout < < "\n";
}

void  sortArray(int intArr[],int n)
{
  int offset,temp,inOrder;
  offset = n;
  while (offset > 1) {
    offset /= 2;
    do {
      inOrder= TRUE;
      for (int i= 0,j= offset;i < (n- offset ); i++ , j++ ) {
        if (intArr[i]> intArr[j]) {
          inOrder = FALSE;
          temp = intArr[i]
          intArr[i]= intArr[j]
          intArr[j]= temp;
```

```
            }
          }
        } while(! inOrder);
    }
}

main()
{
  int arr[MAX]
  int n;

  n= obtainNumData();
  inputArray(arr, n);
  cout < < "Unordered array is:\n";
  showArray(arr, n);
  return 0;
}
```

第 8 天

测　验

1. **枚举值 on 和 off 分别对应不同的枚举类型。正确的语句是:**

```
enum Boolean {false,true};
enum State {state- on,state- off};
enum YesNo {yes, no};
enum DiskDriveStatus{drive- on,drive- off};
```

2. **枚举类型 YesNo 的声明是正确的。**

3. **以下的 IntArrStruct 结构供参考:**

```
struct intArrStruct {
int * dataPtr;
unsigned size;
};
```

4. **以下的 matStruct 结构供参考:**

```
struct matStruct {
double* dataPtr;
unsigned rows;
unsigned columns;
};
```

5. **程序在 return 语句前少一条 delete 语句,以下是正确的程序:**

```
# include< iostream. h>
main()
{
int * p= new int;
cout < < "Enter a number : ";
```

```
    cin >> *p;
    cout << "The square of " << *p << " = " << (*p * *p);
    delete p;
    return 0;
    }
```

练　习

以下的 PTR6.CPP 程序供参考:

```
// C++ program that demonstrates pointers to structured types */

#include <iostream.h>
#include <stdio.h>
#include <math.h>

const MAX_RECT = 4;
const TURE = 1;
const FALSE = -1;

struct point {
  double x;
  double y;
};

struct rect {
  point ulc; // upper left corner
  point lrc; // lower right corner
  double area;
  int id;
};

typedef rect rectArr[MAX_RECT];

main()
{
  rectArr r;
  rect temp;
  rect* pr = r;
  rect* pr2;
  double length, width;
  int offset;
  int inOrder;

  for (int i = 0; i < MAX_RECT; i++, pr++) {
    cout << "Enter (X,Y) coord. for ULC of rect. # "
         << i << " : ";
    cin >> pr->ulc.x >> pr->ulc.y;
    cout << "Enter (X,Y) coord. for LRC of rect. # "
         << i << " : ";
    cin >> pr->lrc.x >> pr->lrc.y;
    pr->id= i;
```

```
    length= fabs(pr- > ulc.x - pr- > lrc.x);
    width= fabs(pr- > ulc.y - pr- > lrc.y);
    pr- > area= length* width;
  }
  // sort the rectangles by areas
  offset = MAX- RECT;
  do {
    offset = (8 * offset )/ 11;
    offset = (offset = = 0) ? 1 : offset;
    inOrder= TRUE;
    pr= r;
    pr2= r+ offset;
    for (int i= 0;
         i< MAX- RECT - offset;
         i+ + , pr+ + , pr2+ + )
      if (pr- > area > pr2- > area) {
        inOrder= FALSE;
        temp= * pr;
        * pr= * pr2;
        * pr2= temp;
      }
  } while (! offset = = 1 && inOrder));

  pr = r; //reset pointer
  // display rectangles sorted by area
  for (i= 0;i< MAX- RECT;i+ + , pr+ + )
    printf("Rect # %d has area %5.4lf\n", pr- > id, pr- > area);
  return 0;
}
```

第 9 天

测 验

1. **字符串** s1 **小于字符串** s2, **因而调用函数** strcpy **将产生错误**。
2. **在调用函数** strncpy **时将** MAX **作为第三个变元值, 这确保字符串** s1 **从字符串** s2 **中拷贝** MAX **个字符(不包括** null **终结符)**。

```
# include < iostream.h>
# include < string.h>
const in MAX= 10;
main()
{
  char s1[MAX+ 1];
  char s2[]= "12345678901234567890123456789O";
  strncpy(s1, s2, MAX);
  cout < < "String 1 is " < < s1
       < < "\nString 2 is " < < s2;
  return 0;
}
```

3. 因为变量 s1 中的字符串大于变量 s2 中的字符串，所以这条语句给变量 i 赋一个正值(即 1)。
4. 因为参数取值中包括偏移量，因而调用函数 strcmp 比较的是子串"C++"和"Basic"。因为"C++"大于"Basic"，所以这条语句给变量 i 赋一个正值(即 1)。
5. 不正确。虽然函数的基本思路是合理的，但局部变量的大小需要常数来确定，一种解决方法是使用 MAX_STRING_SIZE 来表示参数 s 的存储大小。

```
int hasNoLowerCase(const char* s)
{
  char s2[MAX_STRING_SIZE+1];
  strcpy(s2, s);
  strupr(s2);
  return (strcmp(s1, s2) == 0) ? 1 : 0);
}
```

另一种解决方法是，利用动态分配内存建立一个动态局部字符串 s，其中存放参数 s 值的一个拷贝，这种方法用到参数 s 的所有值。

```
   int hasNoLowerCase(const char* s)
{
  char * s2 = new char[strlen(s)+1];
  int i;
  strcpy(s2,s);
  strupr(s2);
  // store result in variable i
  i = ((strcmp(s1, s2) == 0) ? 1 : 0);
  delete [] s2; // first delete local dynamic string
  return i;      // then return the result of the function
}
```

练　习

1. 以下的 strlen 函数供参考：

```
int strlen(const char* s)
{
  int i= 0;
  while (s[i]! = '\0')
    i++;
  return i;
}
```

2. 这是函数 strlen 的另一版本：

```
  int strlen(const char* s)
  {
    char * p= s;
    while (p++ ! = '\0')
      /* do nothing */;
    return p - s;
  }
```

3. 以下的STRING5.CPP程序供参考:

```
# include < stdio. h>
# include < string. h>

main()
{
   char str[]= "2* (X+ Y)/(X+ Z) -  (X+ 10)/(Y-5)";
   char strCopy[41];
   char* tkn[3]= {"+ - * / () ","( ) ", "+ - * / "};
   char* ptr;

   strcpy(strCopy, str); // copy str into strCopy
   printf("%s\n", str);
   printf("Using token string %s\n",tkn[0]);
   // the first call
   ptr= strtok(str,tkn[0]);
   printf("String is broken into: %s",ptr);
   while (ptr) {
     printf(" , %s",ptr);
     // must make first argument a NULL character
     ptr= strtok(NULL,tkn[0]);
   }

   strcpy(str,strCopy);//restore str
   printf("\nUsing token string %s\n",tkn[1]);
   // the first call
   ptr= strtok(str,tkn[1]);
   printf("String is broken into: %s", ptr);
   while (ptr) {
      printf(" , %s",ptr);
      //must make first argument a NULL character
      ptr= strtok(NULL,tkn[1]);
   }

   strcpy(str,strCopy); //restore str
   printf("\nUsing token string %s\n",tkn[2]);
   // the first call
   ptr= strtok(str,tkn[2]);
   printf("String is broken into: %s",ptr);
   while (ptr) {
     printf{"  ,%s", ptr);
     // must make first argument a NULL character
     ptr= strtok(NULL,tkn[2]);
   }
   printf("\n\n");
   return 0;
}
```

第 10 天

测　验

1. 函数定义如下:

```
double factorial(int i)
{ return (i > 1) ? double(i) * factorial(i - 1) : 1;}
```

2. 粗略一看,虽然有些古怪,函数好象还是正确的。case 标号可以迅速提供在 0 到 4 之间的取值所对应的结果。但是,default 语句将截获所有大于 4 的数以及所有负数值,而用负数值进行递归将耗尽内存资源。解决的方法是,当变元值为负数时返回一个非常大的负数值。以下是正确定义的函数:

```
double factorial(int i)
{
  if (i> - 1)
    switch(i) {
      case 0:
      case 1:
        return 1;
        break;
      case 2:
        return 2;
        break;
      case 3:
        return 6;
        break;
      case 4:
        return 24;
        break;
      default:
        return double(i) * factorial(i-1);
    }

  else
    return - 1.0e+30,//numeric code for bad argument
}
```

3. 函数 Fibonacci 的非递归定义如下:

```
double Fibonacci(int n)
{
  double Fib0= 0;
  double Fib1= 1;
  double Fib2;

  if (n= = 0)
    return 0;
  else if (n= = 1     n= = 2)
    return 1;
```

```
  else
    for (int i= 0;i < = n;i+ + ) {
      Fib2= Fib0+ Fib1;
      Fib0= Fib1;
      Fib1= Fib2;
    }
  return Fib2;
}
```

4. **正确。第一个函数使用了正规引用参数,而第二个函数用的是指针参数。**

练　习

以下的 ADVFUN9.CPP **程序供参考:**

```
/*
    C+ +  program that uses pointers to functions to implement a
    linear regression program that supports temporary
    mathematical transformations.
* /

# include < iostream.h>
# include < math.h>

const usigned MAX_ SIZE =  100;

typedef double vector[MAX_ SIZE];

struct regression {
  double Rsqr;
  double slope;
  double intercept;
};

//declare array of function pointers
double (* f[2])(double);

//declare function prototypes
void initArray(double* , double* , unsigned);
double linear(double);
double sqr(double);
double reciprocal(double):
void calcRegression(double* , double* , unsigned, regression&,
                    double (* f[2])(double));
int select_ transf(const char* );

main()
{
  char ans;
  unsigned count;
  vector x, y;
  regression stat;
  int trnsfx, trnsfy;
```

```
  do {
     cout < < "Enter array size [2. ."
          < < MAX- SIZE < < "] : ";
     cin > > count;
  } while (cout < = 1     count > MAX- SIZE);
  // initialize array
initArray(x, y, count);
  // transform data
  do (
  // set the transformation functions
  trnsfx= select- transf("X");
  trnsfy= select- transf("Y");
  // set function pointer f[0]
  switch(trnsfx) {
     case 0:
       f[0] = linear;
       break;
     case 1:
       f[0] = log;
       break;
     case 2:
       f[0] = sqrt;
       break;
     case 3:
       f[0] = sqr;
       break;
     case 4:
       f[0] = reciprocal;
       break;
     default:
       f[0] = linear;
       break;
  }
  // set function pointer f[1]
  switch (trnsfy) {
     case 0:
       f[1] = linear;
       break;
     case 1:
       f[1] = log;
       break;
     case 2:
       f[1] = sqrt;
       break;
     case 3:
       f[1] = sqr;
       break;
     case 4:
       f[1] = reciprocal;
       break;
     default:
```

```
        f[1] = linear;
        break;
    }
    calcRegression(x, y, count, stat, f);
    cout << "\n\n\n"
         << " R-square = " << stat.Rsqr << "\n"
         << "Slope = " << stat.slope << "\n"
         << "Intercept = " << stat.intercept << "\n\n\n";
    cout << "Want to use other transformations? (Y/N) ";
    cin >> ans;
    } while (ans == 'Y'    ans == 'y');
return 0;
}

void initArray(double* x, double* y, unsigned cout)
// read data for array from the keyboard
{
    for (unsigned i= 0;i< count; i++,x++,y++) {
      cout << "X["<< i << "] : ";
      cin >> *x;
      cout << "Y["<< i << "] : ";
      cin >> *y;
    }
}

int select_transf(const char* var_name)
// select choice of transformation
{
  int choice = -1;
  cout << "\n\n\n";
  cout << "select transformation for variable " << var_name
       << "\n\n\n"
       << "0) No transformation\n"
       << "1) Logarithmic transformation\n"
       << "2) Square root transformation\n"
       << "3) Square transformation\n"
       << "4) Reciprocal transformation\n";
  while (choice < 0    coice > 4) {
    cout << "\nSelect choice by number:";
    cin >> choice;
  }
  return choice;
}

double linear(double x)
{ return x;}

double sqr(double x)
{ return x* x;}

double reciprocal(double x)
{return 1.0/x;}
```

```
void calcRegression(double* x,
                    double* y,
                    unsigned cout,
                    regression &stat,
                    double (* f[2])(double))
{
    double meanx, meany, sdevx, sdevy;
    double sum = (double) count, sumx = 0, sumy = 0;
    double sumxx = 0, sumyy = 0, sumxy = 0;
    double xdata, ydata;

    for (unsigned i= 0; i< count;i+ + ) {
      xdata= (* f[0])(* (x+ i));
      ydata= (* f[1])(* (y+ i));
      sumx + = xdata;
      sumy + = ydata;
      sumxx + = sqr(xata)
      sumyy + = sqr(ydata);
      sumxy + = xdata * ydata;
    }

    meanx= sumx/sum;
    meany= sumy/sum;
    sdevx= sqrt((sumxx-sqr(sumx)/sum)/(sum- 1.0));
    sdevy= sqrt((sumyy-sqr(sumy)/sum)/(sum- 1.0));
    stat.slope= (sumxy-meanx* meany* sum)/
                sqr(sdevx)/(sum-1);
    stat.intercept= meany- stat.slope* meanx;
    stat.Rsqr= sqr(sdevx/sdevy* stat.slope);
}
```

第 11 天

测　验

1. 缺省情况下,类成员是保护的。因此,这个类的声明中没有公有成员,不能用于建立实例。
2. 第 3 个构造函数有一个缺省变元,这就使第 4 个构造函数成为多余的。C+ + 编译器能检测到这样的错误。
3. 正确。字符串("Hello Visual C+ + ")建立类的一个临时实例,然后将其赋给实例 s。
4. 正确。新语句是合法的。

练　习

参考程序 CLASS7.CPP 中使用的 main 函数如下:

```
main()
{
  Complex c[5];
```

```
    c[1].assign(3,5);
    c[2].assign(7,5);
    c[4].assign(2,3);

    c[3] = c[1] + c[2];
    cout << c[1] << " + " << c[2] << " = " << c[3] << "\n";
    cout << c[3] << " + " << c[4] << " = ";
    c[3] += c[4];
    cout << c[3] << "\n";   return 0;
}
```

第 12 天

测　验

1. **不正确。**read **和** write **函数不能存储和调出由一个结构或类指针成员存取的动态数据。**
2. **正确。**
3. **正确。**
4. **不正确。**

练　习

以下是程序 IO4.CPP **中的成员函数** binSearch **和更新后的函数** main **的代码。(输出中还显示了新的全局常量** NOT_FOUND **和更新后的类声明):**

```
const unsigned NOT_FOUND = 0xffff;
class VmArray
{
  protected:
  fstream f;
  unsigned size;
  double badIndex;

  public:
    VmArray(unsigned Size, const char* filename);
    ~VmArray()
      { f.close(); }
    unsigned getSize() const
    { return size; }
    boolean writeElem(const char* str,unsigned index);
    boolean readElem(char* str,unsigned index);
    void Combsort();
    unsigned binSearch(const char* search);
};

unsigned VmArray::binSearch(const char* search)
{
  unsigned low = 0;
  unsigned high = size - 1;
```

```
  unsigned median;
  char str[STR_SIZE+1];
  int result;
  do {
    median=(low+high)/2;
    readElem(str,median);
    result=strcmp(search,str);
    if (result>0)
      low=median+1;
    else
      high=median-1;
    else
      high=median-1;
   } while (result!=0 && low<=high);
   return (result==0)?median:NOT_FOUND;
}

main()
{
  const unsigned NUM_ELEMS=10;
  char *data[]={"Michigan","California","Virginia","Maine",
                "New York","Florida","Nevada","Alaska",
                "Ohio","Maryland"};
  VmArray arr(NUM_ELEMS,"arr.dat");
  char str[STR_SIZE+1];
  char c;
  unsigned index;
  //assign values to array arr
  for (unsigned i=0;i<arr.getSize();i++) {
    strcpy(str,data[i]);
    arr.writeElem(str,i);
  }
  //display unordered array
  cout<<"Unsorted arrays is:\n";
  for (i=0;i<arr.getSize();i++) {
    arr.readElem(str,i);
    cout<<str<<"\n";
  }

  //pause
  cout<<"\nPress any key and then Return to sort the array...";
  cin>>c;
  //sort the array
  arr.Combsort();
  //display sorted array
  cout<<"Sorted arrays is:\n";
  for (i=0;i<arr.getSize();i++) {
    arr.readElem(str,i);
    cout<<str<<"\n"
  }
  //pause
  cout<<"\nPress any key and then Return to search the array...";
```

```
    cin > > c;
    // search for array elements using the pointer data
    for (i= 0;i < NUM- ELEMS;i+ + ) {
      index= arr.binSearch(data[i]);
      if (index ! = NOT- FOUND)
        cout < < "Found" < < data[i]
             < < "at index" < < index < < "\n";
      else
        cout < < "No match for " < < data[i] < < "\n";
    }
    return 0;
}
```

第 13 天

测　验

1. **消息框要求如下的** MB- xxxx **表达式**:

 MB- YESNOCANCEL　　MB- ICONQUESTION

2. **消息框要求如下的** MB- xxxx **表述**:

 MB- OK　　MB- ICONEXCLAMATION

3. MessageBox **函数返回** IDYES。
4. MessageBox **函数返回** IDCANCEL。

第 14 天

测　验

1. **正确**。
2. **不正确, 相关的函数是** OnActivate, **而不是** Onactivate。
3. **正确**。
4. **正确**。

第 15 天

测　验

1. **不正确。函数** OnPaint **也初始绘制窗口。当应用程序接收到** Windows **消息** WM- PAINT **时, 窗口会被绘制或重绘**。
2. **不正确。继承的滚动函数没有提供滚动特性**。
3. **不正确。仅当窗口类滚动其内容时才需要成员函数** OnSize。**这时, 运行时系统给** OnSize **提供更新后的窗口尺寸**。
4. **不正确。函数** OnKeyDown **只是禁止使用光标控制建, 如** PageUp **和** PageDown **等**。

第 16 天

测　验

1. 不正确,只有具有 SS_SIMPLE 风格的静态控制文本才不可改变。
2. 正确。
3. 不正确,至少在删除静态控制的动态实例时是需要指针的。
4. 正确。
5. 正确,因为每个控制都是一个窗口。
6. 不正确,由下压按钮发出的 Windows 消息应该由 ON_BTN_CLICKED 映射宏进行映射。

第 17 天

测　验

1. 不正确,只有在两个按钮的选择不相容时,选中框才可以替换它们。
2. 正确。
3. 正确,每个选中框都可以独立地切换。
4. 不正确,分组控制中的圆按钮是相互排斥的。

第 18 天

测　验

1. 正确。
2. 正确。
3. 不正确,列表框通知消息 LBN_CHANGE 表明选中一个新项。
4. 不正确,列表框通知消息 LBN_DBLCLK 表明用鼠标双击选中一列表项。
5. 不正确,LBS_STANDARD 风格中包括了 LBS_SORT 风格,因而,使用它将建立一个排序列表框。

第 19 天

测　验

1. 正确。
2. 不正确。正确的通知消息为 CBN_EDITUPDATE。
3. 不正确,列表框组元中的项是排序的,但不是唯一的,可以插入同一字符串的多个拷贝。
4. 正确,尽管插入列表框组元中的项不是按年代顺序,但它们仍有一定的顺序。
5. 不正确。

6. 正确。程序说明了如下概念: 当改变两个控制中的任一个值, 另一个控制也将随之改变。
7. 不正确, 程序 CTLSCRL1.CPP 表明: 可以在运行时改变一个滚动条的取值范围。

第 20 天

测　验

1. 不正确。可以在编译.CPP 文件后再编译.RC 文件。
2. 不正确。OK 和 Cancel 按钮是必须的。
3. 正确, 但是在程序中这些按钮必须当 OK 和 Cancel 按钮处理。换句说, 可以改变标准的 OK 和 Cancel 按钮的标题。
4. 不正确。嵌套对话框的最好例子是有关打印机设置的对话框。
5. 不正确。对话框可用作独立的窗口。

第 21 天

测　验

1. 不正确。
2. 不正确。
3. 不正确。
4. 正确。最后祝贺你完成本书的学习!

词 汇 表

地址(address):计算机内存中每个内存字节所在的位置。最低的内存地址是 0,最高的内存地址取决于计算机的内存数量。

祖先类(ancestor class):在类层次结构中比所引用的类高一层或多层的类。

变元(argument):传递给函数或过程的参数的值。

数组(array):具有相同名字并使用一个或多个索引访问各个成员的变量的集合。

ASCII 码(ASCII):美国信息交换标准代码(American Standard Code for Information Interchange 的缩写)。

ASCII 文件(ASCII file):包含可读文本的文件。ASCII 文件也称为文本文件。

AUTOEXEC.BAT:特殊的批处理文件。系统初启时,如果计算机在启动盘的根目录下发现它就首先自动执行它。

备份文件(backup file):文件的复制副本,可防止主文件中的文本和数据的损坏。

.BAK:备份文件的常用扩展名。

基类(base class):类层次结构的根。基类没有父类。

批处理文件(batch file):包含 MS-DOS 命令和其他特殊命令的可执行文件。批处理文件有助于实现多个任务的自动化。

二进制(binary):只包含 0 和 1 的数制系统。

位(bit):二进制数字(binary digit)的略名。位是数据存储的最小单位。程序中常用 8、16、32 和 64 位组来代表不同种类的数据。

布尔(Boolean):与“逻辑”(真或假)同义。布尔一词来源于对逻辑值和真值表作出研究的英格兰数学家乔治·布尔(George Bool)。

冒泡排序(bubblesort):将数组元素按升序或降序排列的最慢且最简单的方法。

缓冲区(buffer):供临时数据流进流出的内存区。

臭虫(bug):程序的逻辑错误的通俗称谓。(在中文中一般转译为“错误”。)

字节(bytes):包含 8 个数位的数据存储单位。

.C:C 文件的扩展名。

CGA:彩色图形适配器(Color Graphics Adapter)的缩略词。CGA 屏幕的分辨率是 640×200 像素点。

类(class):对象的种类。

剪贴板(clipboard):Windows 环境中保存拷贝或删除的文本的特殊内存区。

时钟滴答(clock tick):以 CPU 内部时钟为基准的时间长度。1 秒钟内有 18.2 次滴答。(tick 是象声词,也可转译为“计时”。)

梳形排序(Comb sort):对数组元素排序的新的有效方法。这种方法基于改进的冒泡法。

编译器(compiler):将程序文件中的源代码转换成二进制目标代码(.OBJ)文件的工具。目标代码必须与其他目标文件和库文件链接才可生成可执行的.EXE 文件。

连接(concatenation):将两个或多个串中的字符接在一起,从而将它们连接起来的过程。

条件循环(conditional loop):测试条件满足时即停止迭代的循环。

CONFIG.SYS:系统配置文件的名字。机器启动时要读出位于启动盘的根目录下的 CONFIG.SYS 文件。

常量(constant):在程序执行期间固定不变的量。

.CPP:C++ 源文件的扩展名。

CPU:中央处理器(Central Processing Unit)的缩略词,计算机的大脑。

死机(crash):计算机好像冻结且对键盘、鼠标或任何其他输入设备都没有响应的一种状态。

光标(cursor):出现于屏幕上的闪烁的下划线或方块,用以指示下一个字符插入之处。

数据成员(data member):类中存储数据的域。

数据处理(data processing):对信息的操作与管理。

数据记录(data record):表达有某种意义的信息的数据项的集合。例如,通信地址就是一个数据记录。

调试(debug):检测并去掉程序中的逻辑错误。(其实这个词的本意便是“去臭虫”。)

调试器(debugger):辅助调试程序的工具。C++ 的集成环境中就包含一功能强大的调试器。

.DEF:Windows 程序中约定的定义文件的扩展名。

缺省(default):一个预选的动作或值,当未提供其他动作或值时就采用它。

子孙类(descendant class):从父类派生出的任意子类。

对话框(dialog box):提示你进行输入的特殊的弹出画面。

目录(directory):硬盘、软盘或电子盘中包含有文件的片段或部分。

磁盘(disk):存储数据的磁介质。

磁盘驱动器(disk drive):读取并装载磁盘上的数据和程序的设备。

软盘(diskette):可移动的薄磁盘。也称为软磁盘(floppy disk)。

显示器(display): 计算器的屏幕、控制台或监视器。

显示适配器(display adapter): 计算机内辅助显示文本和图形的电路板。

DOS: 磁盘操作系统(Disk Operating System)的缩略词。

拖曳(drag): 按住鼠标按钮移动鼠标的动作。此动作往往可以将窗口从屏幕上的一个位置移到另一位置。

EGA: 增强型图形适配器(Enhanced Graphics Adapter)的缩略词。EGA 显示器的分辨率是 640× 350 像素点。

(数组的) 元素(element): 数组的成员, 可由一个或多个索引访问。

文件(file): 磁盘上包含有程序、文本或其他数据的场所的名字。

文件扩展名(file extension): DOS 文件用来标识文件类型的三个字母。可执行的程序的文件扩展名是. COM 或. EXE。

文件 I/ O(file I/ O): 文件输入与输出。

文件名(filename): DOS 文件的名字。

格式(format): 用来输出数据的映像或图。

函数(function): 执行一条或多条语句并返回某个值的子程序。

功能键(function key): 键盘上标着 F1 到 F10 的键。有些键盘还有标着 F11 和 F12 的键。

全局变量(global variable): 可被程序中的所有函数和过程访问的变量。

图形监视器(graphics monitor): 能显示高分辨率图形的监视器。

. H: 头文件的扩展名。

头文件(header file): 包含有常量声明、数据类型、函数原型以及类声明的文件。

赫兹(hertz): 用来度量频率的单位。1 赫兹就是每秒 1 个周期, 简写为 Hz。

HGA: 大力神图形适配器(Hercules Graphics Adapter)的缩略词。

继承(inheritance): 一个类可以复用其父类中的数据成员和成员函数的能力。

输入(input): 程序从键盘、磁盘驱动器、通信端口或其他任何输入设备中所获取的数据。

I/ O: 输入/ 输出(Input/ Output)的缩略词。

千字节(kilobyte, K): 度量存储器的单位。1K 是 1024 个字节。

最低位(least significant bit): 字节、字或任意其他内存单位的最右边的一位。

链接程序(linker): 一种工具, 它可以将编译器生成的不同的. OBJ 文件和不同的库文件组装起来以生成可执行的. EXE 文件。

局部变量(local variable): 在函数或过程内部声明的变量。

循环(loop):重复执行的语句集。

主模块(main module):包含有函数 main 的程序部分。

.MAK:项目生成文件(make file)的扩展名。

生成文件(makefile):根据需要调用编译器和链接程序,用以编译和链接源文件与目标文件的特殊文件。

数学运算符(math operator):进行数学运算,如加、减、乘、除以及乘方的符号。

矩阵(matrix):二维数组。矩阵的元素须通过指定行和列索引来访问。

MCGA:多彩色图形适配器(Multi-Color Graphics Adapter)的缩略词。

MDA:单色显示适配器(Monochrome Display Adapter)的缩略词。

兆字节(megabyte,M):存储器的单位。1M 约 1 百万字节。

成员函数(member function):作为类成员的函数。

内存(memory):计算机在开机状态时,机器内存放程序和数据的部分。(根据不同场合,有时也译作"存储器"。)

菜单(menu):提供可供选择的多个任选项显示画面。

菜单驱动(menu-driven):在用户指导下依靠一个或多个菜单以完成其任务的程序。

消息(message):消息是由面向对象的程序发送给对象的。

方法(method):用于描述面向对象的程序如何操作对象。

模(modulus):除法的整数余数。

监视器(monitor):屏幕、控制台或显示器。

单色(monochrome):单一色彩。单色监视器既不能显示彩色又不能显示高分辨率图形。

最高位(most significant bit):字节、字或其他内存单位的最左边一位。

MS-DOS:Microsoft 磁盘操作系统(Microsoft Disk Operating System)的缩略词。

多维数组(mutidimensional arrays):有多个下标的数组。

嵌套循环(nested loop):在另一循环之内的循环。

空串(null string):空字符串。

数值函数(numeric functions):返回数值的函数。

对象(object):类的一个实例。

目标文件(object file):编译器的输出。链接程序将不同的目标文件和库文件组装起来以生成可执行的.EXE 文件。

面向对象编程(OOP,object-oriented programming):强调建立对象及其操作的模型的现

代编程技术。

开循环(open loop):理论上无限迭代的循环。实际上,大多数开循环都使用位于循环内部的语句退出。

运算符顺序(order of operators):表达式中执行运算的优先级。

输出设备(output device):接收输出的设备。常用的输出设备是屏幕、打印机、磁盘驱动器以及通信端口。有些输出设备也支持输入,如通信端口和磁盘驱动器。

调色板(palette):可用的颜色集。

参数(parameter):出现在函数或过程的声明后的特殊变量,它可微调函数或过程的操作。

引用传参(passing by refernce):将变元的引用作为参数传递给函数或过程的操作。结果,接收引用的参数变成了变元的别名,在函数或过程内部对参数作的任何改变都会对该变元产生影响。

数值传参(passing by value):将变元的数值副本作为参数传递给函数或过程的方法。结果,在函数或过程内部对参数所作的任何改变都不会影响到该变元。

像素(pixel):图形像素(picture element)的简称。

指针(pointer):存储另一变量或系统信息的特殊变量。

多态性(polymorphism):面向对象程序设计的特性之一,它可支持抽象功能。

运算符的优先级(precedence of operators):见"运算符顺序(order of operators)"。

程序(program):可执行的语句和指令的集合。

程序设计语言(programming language):用来与机器交流的高级用户接口。

伪代码(pseudocode):用英语描述程序任务的文字纲要。

RAM:见"随机存取存储器(random access memory)"。

随机存取文件(random access file):允许通过指定记录号从而访问其定长记录,从而用于输入和输出的文件。随机存取文件适用于数据库。

随机存取存储器(RAM, random access memory):通过指定地址进行访问的存储器。

.RC:资源描述文件的扩展名

只读存储器(ROM, read only memory):一种包含着固定数据的存储器。

记录(record):信息在数据文件中的存储单位。一般来说,文本文件含变长记录,而随机存取文件含定长记录。

关系运算符(relational operators):比较两个相关数据项并返回一逻辑值的符号。

.RES:编译过的资源文件的扩展名。

分辨率(resolution):屏幕上所显示的图像的清晰度。

资源文件(resource file):用以定义用于一个或多个 Windows 应用程序中的资源的文件。资源文件可以被不同的程序设计语言编写的应用程序所共享。资源包括串、键盘、菜单、控制和对话框资源。

ROM:见"只读存储器(read only memory)"。

科学计数(scientific notation):用尾数和指数表示数的一种特殊形式。例如,在 1.23E+44 这个数中,1.23 是尾数而+44 是指数(10 的幂)。可见,用 1.23E+44 标识 1.23 x 10 + 44 更方便一些。

顺序文件(sequential file):使用变长记录存储可读文本的文件。

一维数组(single-dimensional arrays):用一个下标访问其元素的数组。

排序(sorting):用升序或降序排列数组元素的操作。数组可按各元素的部分或全部值排序。

静态变量(static variable):具有固定内存位置的变量。通常,静态变量在函数中定义并在函数调用之间延续其值。

串常量(string constant):与一个串文字相关联的常量。

串文字(string literal):括在一对双引号之内的字符集。

串变量(string variable):保存字符串的变量。

结构(structure):用户定义的数据类型。C++ 中允许用 TYPE 语句定义结构。

结构化数组(structured array):包含有用户定义的数据类型的元素的数组。

结构化变量(structured variable):其类型是用户定义的数据类型的变量。

子目录(subdirectory):与其父目录相连的目录。

下标(subscript):数组的索引。

SVGA:超级向量图形阵列(Super Vector Graphics Array)。参阅"VGA"。

语法错(syntax error):写语句时出的错。

系统挂起(system hang):见"死机(crash)"。

文本文件(text file):见"ASCII 文件(ASCII file)"。

单目运算符(unary operator):仅需一个操作数的运算符。

用户定义函数(user-defined function):由程序员定义的函数,它从概念上对 C++ 语言进行扩展并服务于主程序。

变量(variable):存储数据的且带有标记的内存位置。

变长记录(variable-length record):具有可变数量的字节或字符的记录。

变长串变量(variable-length string variable):可容纳可变数量字符的串变量。

变量作用域(variable scope):**变量在程序的各个部分的可见性**。

VGA:**向量图形阵列**(Vector Graphics Array)**的缩略词**。VGA **的分辨率是** 640× 480 **像素点**。

虚函数(virtual function):**一种函数,当它在子孙类中被重写时,它的用途和参数表必须与祖先类中的相同**。

Windows **消息**(Windows message):Windows **的一个部分发送到另一部分的消息**。

字(word):**包含** 2 **字节或** 16 **位的数据存储单位**。